INITIALIZATION STRATEGY AND ACTIVATION FUNCTION SELECTION FOR NEURAL NETWORKS BASED ON GAUSSIAN PROCESS OPTIMIZATION

Anthony S. Tai

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Department of Statistics,

Indiana University

May, 2021

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Chunfeng Huang, Ph.D.

David Crandal, Ph.D.

Michael Trosset, Ph.D.

Daniel Manrique-Vallier, Ph.D.

Andrew Christianson, Ph.D.

May, 2021

This thesis work is dedicated to my lovely wife, Kellie, my awesome daughter, Kaelyn, my brother, Roger, my sister, Connie, and in loving memory, to my parents.

Acknowledgments

I would first like to thank the United States Navy Naval Surface Warfare Center Crane Division leadership and Ph.D. Fellowship Selection Committee for the opportunity to pursue a dream that did not seem possible. I am grateful for the generous financial support provided by the Crane Ph.D. Fellowship Program.

It has been a challenging journey. It would have been even more demanding without my advisor's continual guidance. Dr. Huang taught me not only how to do research but also what doing research is about. I am thankful for his patience and encouragement. I would also like to thank my committee members: Dr. Crandall, Dr. Trosset, Dr. Manrique-Vallier, and Dr. Christianson, for their valuable time and feedback. As I struggled with research and preparation for dissertation, Dana and Kelly in the Statisticd Department made sure I kept up with my program schedule. I sincerely appreciate their help.

I owe a debt of gratitude to my family. It would be an understatement to say that they helped keep me going in this venture. My brother, Roger, and my sister, Connie constantly cheered me on with their moral support. Always a patient listener and a good public speaker, my daughter, Kaelyn, inspired me to express important ideas in plain language, precisely and concisely. Finally, my ever-patient wife, Kellie, was always there to lift me up when I was discouraged, and share my excitement when things went in the right direction. I promised her that I will "take a break" after this endeavor. You have my word, my love!

Anthony S. Tai

INITIALIZATION STRATEGY AND ACTIVATION FUNCTION SELECTION FOR NEURAL NETWORKS BASED ON GAUSSIAN PROCESS OPTIMIZATION

To achieve better prediction performance, much research effort in deep/machine learning has been devoted to improving neural network model development and training. Neural networks have been implemented in healthcare systems, image classification, navigation and exploration applications. However, neural network models, in general, may fail to train effectively without proper preparation. In view of this, choosing appropriate initialization schemes and activation functions remains an important research topic within the deep learning and machine learning communities. My dissertation studies the connection between Gaussian processes and neural networks. It seeks to leverage their synergies to enhance neural network initialization and activation function selection for improved model accuracy.

In my dissertation I investigate marginal likelihood maximization, a Gaussian process procedure that learns from data. Prediction tasks on real-world and simulated data are performed with networks initialized with learned hyperparameters. The objective is to evaluate the statistical technique for assisting model initialization in single- and multi-layer neural networks. Furthermore, a simulation is carried out to assess the method for activation function selection in single-hidden-layer neural networks. Empirical results suggest that the proposed Gaussian process technique is a promising approach for guiding neural network model initialization and activation function selection to achieve improved prediction performance.

There are three main contributions in this dissertation. First, I investigate the link between neural networks and Gaussian processes. Second, I implement and validate the method of marginal likelihood maximization for improving initialization and model prediction in single- and multi-layer neural networks. Lastly, I demonstrate that under certain conditions the Gaussian process technique is also effective in selecting activation functions in neural network models.

Chunfeng Huang, Ph.D.

David Crandal, Ph.D.

Michael Trosset, Ph.D.

Daniel Manrique-Vallier, Ph.D.

Andrew Christianson, Ph.D.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Importance of Neural Network Initialization and Activation Function

Advanced research in structural design and algorithmic development in network prediction models have led to many successes in machine learning applications. Researchers developed a navigation system using machine learning technique [1] to generate rapid 3D spatial awareness for navigation and exploration. Researchers also designed a machine learning model [2] to estimate the seriousness of a patient's pulmonary edema, a condition describing excess fluid in the lungs, by examining chest radiographs. To achieve better neural network prediction, much research effort has been devoted to improving model training.

Developing a good prediction model requires designing proper initialization schemes and choosing appropriate activation functions. Researchers have proposed various techniques to improve model initialization, such as the normalized initialization (aka Xavier-initialization) in [3], He-initialization in [4], layer-sequential unit-variance (LSUV) initialization in [5], an initialization strategy for noisy ReLU networks in [6], and the Gaussian SubMatrix method that makes use of initial parameter sharing in [7]. Moreover, Ramachandran et al. [8] and Hayou et al. [9] demonstrate that selection of activation function in a neural network plays a critical role in achieving good model accuracy. These are just a few examples supporting the notion that neural network initialization and activation function selection remain important research topics within the deep learning and the machine learning communities.

1.2 Research Goal and Approach

This dissertation seeks a data-driven statistical approach for neural network initialization and activation function selection to improve model training and prediction accuracy. Intuitively, a machine learning model built with prior knowledge about the data on which it makes predictions should perform more accurately than without. This leads me to explore Gaussian process optimization for learning relevant information from training data. To utilize information obtained from a Gaussian process model to guide neural network initialization and activation function selection, I research the link between the models' structures. In particular, Neal in [10] establishes the equivalence of a fully-connected, single-hidden-layer neural network and a Gaussian process. He shows that in convergence the prior joint distribution of an infinitely wide network output values is a Gaussian process. This results in an expression relating the activation function of the neural network and the covariance function of the Gaussian process.

Although Neal proves the equivalence of the models, he does not provide the exact form of any covariance function. However, in [11] Cho and Saul develop closed-form representations for arc-cosine kernels/covariance functions corresponding to one-sided polynomial activation functions. I mainly follow Cho and Saul's procedure to derive an alternative form of covariance function corresponding to rectified linear unit (ReLU) activation function.

Given the covariance function, I compute the marginal likelihood [12] of the Gaussian process to measure the probability of the model given the observed data. Marginal likelihood maximization tends to choose the model that is best for explaining the data. Once the model is chosen, hyperparameters that characterize the model are extracted to initialize the neural network. In principle, neural networks initialized with these hyperparameters should achieve improved accuracy. This initialization scheme is termed GP-init in this dissertation as it results from using a Gaussian process model.

1.3 Research Experiments

To assess GP-init, I design and conduct experiments to evaluate neural network prediction accuracy based on the statistical method. First, neural networks are trained with initial hyperparameters obtained from marginal likelihood maximization. Next, prediction tasks are performed on simulated and real-world datasets with the trained models. Model prediction results are then compared to those obtained from a benchmark initialization method. In addition, experiments are carried out to evaluate the GP-init method for activation function selection in single-hidden-layer neural networks. The details of these experiments are described in Chapter 3 for single-hidden-layer neural network initialization, Chapter 4 for multi-layer model initialization, and Chapter 5 for activation function selection for single-hidden-layer models.

1.4 Outline of the Remainder of the Dissertation

The remaining chapters of this dissertation discuss the relationship between neural networks and Gaussian processes, and the procedure for linking the models to develop a strategy for neural network initialization and activation function selection.

In Chapter 2, I address the issue of learning from data to improve neural network initialization. The structure and properties of a neural network are first described, followed by an introduction to Gaussian process model where log marginal likelihood maximization is demonstrated to learn model hyperparameters. I then describe the equivalence of neural network and Gaussian process based on [10] to develop a neural network model initialization strategy and activation function selection technique.

In Chapter 3, I evaluate the Gaussian process initialization (GP-init) method on a single-hidden-layer, fully-connected feedforward neural networks with rectified linear unit (ReLU) activation function. A statistical method is used to derive the corresponding Gaussian process covariance function. Simulations are then performed to assess GP-init on a neural network regression task.

Evaluation is also done on a real-world image prediction task.

In Chapter 4, I extend the study to multilayer models. Parallel to the experiments performed in Chapter 3, I evaluate the GP-init method on ReLU neural networks with two and three hidden layers, respectively.

In Chapter 5, my main goal is to demonstrate how marginal likelihood maximization can be utilized to select activation function for neural network models. Additionally, in the case when the covariance function is analytically intractable, I show that Monte Carlo Approximation can be used to address this issue.

In Chapter 6, I conclude this dissertation with a summary of contributions and a list of tasks for future research.

Chapter 2

Background

2.1 Neural Networks

A neural network is a prediction model composed of layers of individual processing nodes linked through weighted connections. Its input layer passes the input data to the output layer through a number of hidden layers to complete a task, such as producing an estimate of a target value associated with the input in a regression task. The neural network design specifies its architectural structure, optimization scheme, cost function, and the activation function as the network nonlinearity mapping input to output.

As a prediction model, a neural network is tool for input-output mapping. In fact, it was shown in [13, 14, 15] that under certain conditions a neural network can approximate any continuous function to arbitrary precision. To obtain good prediction results, a neural network model requires proper training. It is usually first trained with pairs of observed input and target output to optimize its training convergence and accuracy. Given an unseen input, the well-trained neural network may then be able to recognize relationships between data to make predictions about the target.

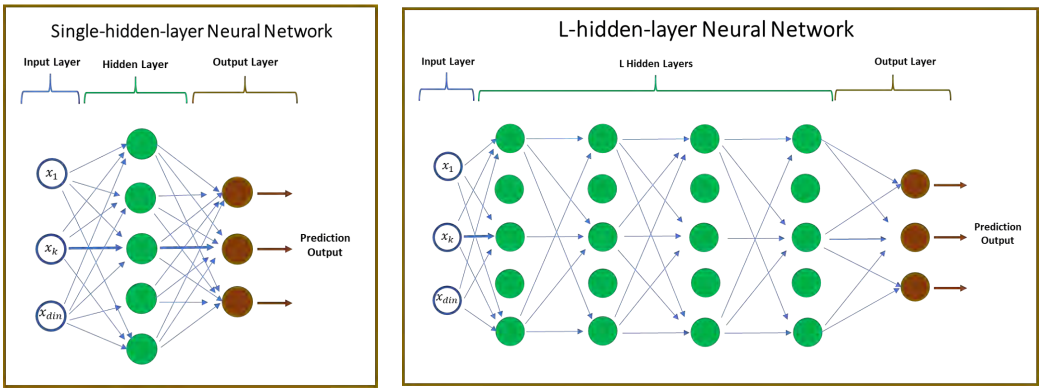

Figure 2.1: Structural diagrams of simple neural networks.

The architecture of the simplest neural network consists of one single hidden layer situated between the input and output layers. A diagram of a fully-connected, feedforward neural network with a single hidden layer is shown in the left panel in Figure 2.1.

Recent development in deep learning discovered that network models with multiple layers of computational nodes are able to extract various levels of abstractions, allowing them to learn representations of complex data [16]. The structural diagram of a fully-connected network with L hidden layers is depicted in the right panel in Figure 2.1. Striving to improve the predictive performance of multilayer networks, researchers investigated how these networks could be trained successfully so that the models converge to acceptable solutions quickly. Functionalities of rectifying processing nodes were studied and activation functions such as rectified linear unit (ReLU) and parametric ReLU (PReLU) were proposed to replace the standard sigmoidal hyperbolic tangent function in deep network applications [17, 4].

2.1.1 Activation functions

Activation functions are functions applied by the computational nodes in hidden and output layers in neural networks to process information traveling through the network. They are typically nonlinear functions to facilitate mapping of complex input data to desired output in the model. There are various activation functions available for machine learning applications [18].

Selection of activation functions is important because inappropriate choice can lead to inadequate model training and poor prediction accuracy. In [3], Glorot and Bengio study the behavior of sigmoid, hyperbolic tangent and softsign activation functions across network layers during model training. They determine that sigmoidal logistic activation function should not be used in conjunction with random initialization in deep networks. He et al. define and describe parametric ReLU (PReLU) activation function in [4]. Their experimental results show improvements over ReLU in classifying ImageNet 2012 dataset when PReLU is used, suggesting the value of adaptive learning

activation functions. Very recently, Hayou et al. [9] study how smoothness in activation functions affects model training. Their empirical results indicate that smooth versions of ReLU including Swish [8] give better prediction than ReLU in deep networks.

Examples of some popular activation functions are shown in Figure 2.2, with ReLU, logistic, and Swish in the top panel, and PReLU, tanh, and softplus in the bottom panel.

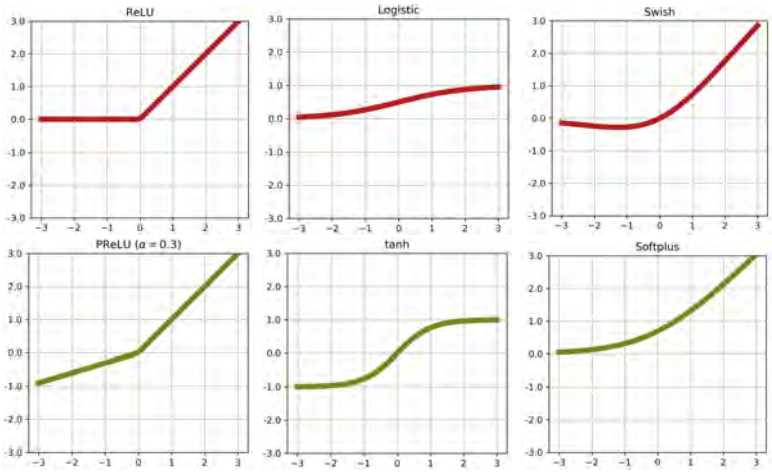

Figure 2.2: Examples of popular activation functions.

2.1.2 Model initialization

Recognizing the difficulty in training multilayer networks, researchers also began tackling the issue of model initialization. Typically the initial values of the model parameters, specifically the weights and biases, are randomly assigned before training occurs.

However, [3] shows that standard gradient descent optimization algorithm from random initialization produces poor prediction results. Their investigation on the variances of layer activations and back-propagated gradients led to the *normalized initialization* (also known as the Xavier-initialization) where the initial bias is assumed to be 0 and the initial weight value W between layer j and layer $j+1$ with layer widths n_j, n_{j+1}, respectively, is uniformly distributed:

$$W \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\Big].$$

Instead of assuming that the network activation functions are linear, [4] develops an alternative approach to model initialization with ReLU/PReLU rectifiers. Assuming that biases are 0 and weights are symmetrically distributed about 0, and anticipating that the variances of input at the first and last layers being equal, the authors propose the He-initialization scheme for the weights between layer l with layer width n_l and layer $l+1$:

$$W \sim \mathcal{N}\Big(0, \frac{2}{n_l}\Big)$$

Researchers have also studied the combined effect of initialization and momentum in deep and recurrent neural networks [19] and conclude that poor initialization likely caused unsuccessful model training. In [5] it was shown that pre-initialization with orthonormal matrices followed by output variance normalization produces prediction performance comparable to, if not better than, standard techniques. Additionally, [20] discusses the bound on the network depth based on the principle of 'Edge of Chaos' given a particular set of initialization hyperparameters. Furthermore, [21] shows that theoretically and in practice proper initialization parameter tuning with appropriate activation function is important to model training for improved performance.

Even though many existing initialization methods have been applied successfully in multilayer network models, they do not necessarily take full advantage of the information in observed data that is relevant to model initialization. I address this issue in my research by exploring statistical methods that learn directly from training data to guide the selection of initial hyperparameters and activation functions in neural networks with the goal of improving model training and prediction accuracy.

2.1.3 Model training and testing

Developing a neural network involves designing the model architecture. Careful consideration must be given to the number of hidden layers, the amount of nodes in each hidden layer, and the activation function that enables the network to learn complex patterns. In addition, optimization

techniques such as stochastic gradient descent and adaptive moment estimation, cost functions like mean-squared-error and binary cross-entropy losses, and performance metric are chosen depending on the task at hand. Lastly, the model needs to be trained in order to perform prediction.

Neural network training is an iterative process. The purpose is to prepare the model so that its prediction performance meets the task requirements in accuracy and stability. The process consists of forward and backward stages. During the forward stage, the network passes the input information through hidden layers to the output layer. Within each hidden layer there are nodes that process the incoming signals using the nonlinearity assigned to them. The output layer then collects the hidden layer results to compute training error. A key component in helping the model to learn during training is to propagate error information back through the network and adjust its parameters to reduce training error. The forward-/back-propagation is repeated with new training data until the training error reaches a design threshold. The model is then ready for performance assessment.

Evaluating a trained neural network involves feeding previously unseen test data to the model and measuring the difference between the predicted output and the ground truth target value. For regression tasks, the metric mean squared error is usually used to compute the numerical difference. On the other hand, classification tasks typically require finding how often predictions and target labels are mismatched.

In this work, I conduct a number of simulations and real-world experiments using neural network models to assess the proposed Gaussian process method for initialization and activation function selection. Simulations are designed to observe the behavior of the GP-init method under various conditions. Real-world experiments on MNIST image dataset is employed for further validation.

Model training in simulation experiments

Each simulation data point is composed of a location and a response value where location $\in \{0.0, 0.01, \cdots, 1.0\}$. A Gaussian process model with ReLU covariance function produces the responses associated with location inputs. The dataset is then split into a training set with 70 points and a test set with the remaining 30 points.

Throughout the course of my dissertation research each neural network model is built with a fully-connected, feedforward architecture. Since all inputs are one-dimensional, it is reasonable to set the neural network hidden layer width to 3. Also, regression tasks require only a single node at the output layer. For discussions on model initialization, ReLU activation function is assigned to each hidden node as the model nonlinearity. For the discussion on activation function selection, neural network models are also constructed with logistic activation for performance comparison. In all cases, the output node simply employs the linear activation function.

For model training, Adam optimizer is applied with mean-squared-error loss and mean-squared-error performance metric. During training the model continues to update its system parameters until the difference between predicted output and the target responses reaches the design requirements.

Once the iterative training process is finished, the model is evaluated with the test set. Upon the completion of testing, the network model produces prediction accuracy results. The entire procedure is repeated 50 times for analysis.

Model training with real-world datasets

The MNIST image dataset [22] is chosen for the real-world experiments. It is a large dataset widely used for neural network training and validation. The database has 60,000 training and 10,000 testing 28x28 pixel gray scale hand-written single digits. Some examples are shown in Figure 2.3. Each image is flattened into a linear vector of 28x28=784 values before being fed to

the model input layer. Sandwiched between the input and output layers are the hidden layers each with 1000 hidden nodes.

Following the empirical study in [23], classifying MNIST images is considered as regression prediction. Inasmuch as the network is designed for regression tasks, the model employs mean-squared-error loss, Adam optimizer, and accuracy as the performance metric for model training. In addition, one-hot encoding is utilized to generate class labels, where an incorrectly labeled class is designated -0.1, and a correctly labeled class 0.9 . For example, the one-hot representation of the integer 7 is given by [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, 0.9, -0.1, -0.1].

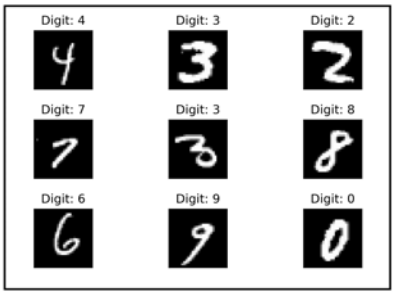

Figure 2.3: MNIST handwritten digit images.

2.2 Statistical Machine Learning

Practitioners in statistical sciences use statistical methods like regression models, analysis of variance, and factorial designs to solve problems [24]. Moreover, exploratory data analysis [25, 26] provides tools for studying the structure of data in detail. As modern technologies continue to advance, the amount of data generated daily in the world is staggering [27]. To decipher and understand data accurately and efficiently, one may turn to statistical machine learning. Statistical machine learning [28, 29] offers numerous models and methods including support vector machines, kernel methods, and graphical models to help extract useful and important information from data. As a result, statistical machine learning models have often been applied to make predictions.

A rather unique probabilistic prediction model, the Gaussian process framework [29] is chosen to be studied in my dissertation research.

2.2.1 Gaussian process basics

A Gaussian process [29, 30, 31, 32] is a set of random variables any finite collection of which follows a multivariate normal distribution. A Guassian process prediction model exploits this unique property and offers a Bayesian approach to solving machine learning problems. The model is completely specified by its mean function and covariance function.

By choosing a particular covariance function, a prior distribution over functions is induced which, together with observed inputs and targets, is then used to generate posterior distribution for making predictions and uncertainty measures on unknown test points. These capabilities allow Gaussian processes to be used effectively in many important machine learning applications such as human pose inference [33] and object prediction [34]. Recent research works also apply Gaussian processes in deep structures for image prediction [35] and regression tasks [36].

2.2.2 Gaussian process prediction: A demonstration

Performing simulations allows us to explore and understand some properties of the models we wish to study. Simulation results also offer the opportunity for evaluating model precision and insight into model behavior.

To describe the procedure of making predictions with a Gaussian process model, I borrow equations from [32] where the formulation of Gaussian process predictive distribution is treated in great detail. Consider a set of N multidimensional input data $X = \{x_i\}_{i=1}^{N}$, $x_i \in \mathcal{R}^D$, and target set $y = \{y_i\}_{i=1}^{N}$, $y_i \in \mathcal{R}$. For each input x_i there is a corresponding input-output pair (x_i, y_i), where the observed output target is given by $y_i = f(x_i) + \epsilon_i$, with data noise $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. I model the input-output latent function f as a Gaussian process :

$$f(x_i) \sim \mathcal{GP}\big(\mu(x_i), c(x_i, x_j)\big),$$

where it is customary to set the mean function $\mu(x_i) \coloneqq E[f(x_i)] = 0$, and denote $c(x_i, x_j)$ as the covariance function.

Consider the unknown test data X_*, and their function values $f_* \coloneqq f(X_*)$. Here the goal is to make predictions of X_*. It is known that the joint distribution of the target and function values is computed as

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X, X_* \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right),$$

where $K(X, X)$ represents the covariance matrix of all pairs of training points, $K(X, X_*)$ denotes that of pairs of training and test points, and $K(X_*, X_*)$ gives the covariance matrix of pairs of test points.

In addition, the prediction distribution is the conditional distribution

$$f_*|X, y, X_* \sim N\big(\mu_*, \Sigma_*\big),$$

$$\text{with mean function } \mu_* = K(X_*, X)\big[K(X, X) + \sigma_n^2 I\big]^{-1} y$$

$$\text{and covariance } \Sigma_* = K(X_*, X_*) - K(X_*, X)\big[K(X, X) + \sigma_n^2 I\big]^{-1} K(X, X_*).$$

A Gaussian process regression task is demonstrated below. The procedure utilizes a square-exponential [37] covariance function

$$k(x, y) = \sigma^2 \exp\big(-\frac{\|x - y\|^2}{2l^2}\big),$$

with the scale factor σ^2 and the lengthscale parameter $l > 0$ to predict target values produced by the latent function $f(x) = \sin(x - 2.5)^2$.

Given the design covariance function $k(x, y)$ and input values, the prior distribution of the Gaussian process model is computed to generate sample paths in the left panel in Figure 2.4. Once the training targets, represented by the red dots in the right panel, are observed the prediction distribution of the Gaussian process is formed to produce alternative solutions. The mean of the distribution is displayed in black. The shaded regions in the panels represent the 95% confidence interval within which a path is generated. Prediction on previously unseen input can then be made using the sample paths.

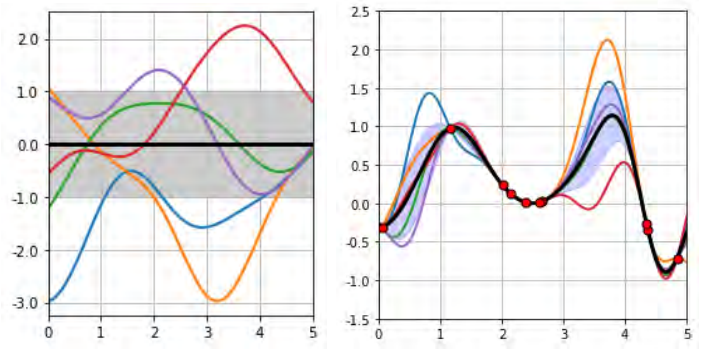

Figure 2.4: Solving a regression problem using Gaussian process.

To help improve performance in Gaussian process prediction, one first need to select a suitable covariance function and then tune the model by adjusting the hyperparameters characterizing the covariance function. These hyperparameters can be estimated through marginal likelihood maximization [12]. This feature enables Gaussian processes to learn proper hyperparameter values from training data and is a key focus of my research.

2.2.3 Marginal likelihood and hyperparameter estimation

I now briefly describe the procedure for estimating optimal hyperparamter values from the data. The marginal likelihood (or evidence) [12, 29] measures the probability of obtaining the observed targets given input data. It can be expressed as the integral of the product of likelihood and the prior, marginalized over the latent function f:

$$p(y|X) = \int p(y, f|X)\, df = \int p(y|f, X)p(f|X)\, df. \tag{2.1}$$

The marginal likelihood can be obtained by either evaluating the integral (2.1) or by noticing $\{y_i\}_{i=1}^N = \{f(x_i) + \epsilon_i\}_{i=1}^N$, which gives us $y|X \sim \mathcal{N}(0, \mathcal{C} + \sigma_n^2 I)$ where $\mathcal{C} = [c(x_i, x_j)]_{i,j=1}^N$ and I are N by N covariance matrix and identify matrix, respectively. As a result,

$$p(y|X) = \frac{1}{(2\pi)^{N/2}|\mathcal{C} + \sigma_n^2 I|^{1/2}} \exp\Big(-\frac{1}{2} y^T (\mathcal{C} + \sigma_n^2 I)^{-1} y\Big).$$

To facilitate computations, I evaluate instead the log marginal likelihood which is given by

$$\log\, p(y|X) = -\frac{1}{2}y^T\left(\mathcal{C} + \sigma_n^2 I\right)^{-1}y - \frac{1}{2}\log\,|\mathcal{C} + \sigma_n^2 I| - \frac{N}{2}\log\,2\pi. \tag{2.2}$$

Recall that the covariance matrix $\mathcal{C}$ is a function of the hyperparameters $\{\sigma_w^2, \sigma_b^2\}$, given fixed training inputs X and targets y. Estimating hyperparamter values for the Gaussian process model is done via maximizing its log marginal likelihood function (Equation 2.2). Grid search can be set up where the log marginal likelihood is computed for specific combinations of $\sigma_w^2, \sigma_b^2 \in \mathcal{G}$. The desired estimates are obtained as

$$\{\tilde{\sigma}_w^2, \tilde{\sigma}_b^2\} = \arg\max_{\sigma_w^2, \sigma_b^2 \in \mathcal{G}} \log\, p(y|X). \tag{2.3}$$

The resulting hyperparameter values become the recommended initial values for the neural network model. My research investigates whether the log marginal likelihood maximization can be an effective method for guiding neural network initialization and activation function selection.

Note that the marginal likelihood is applied on the entire training dataset. In addition, Cholesky decomposition [31] can be employed to calculate the term $\left(\mathcal{C} + \sigma_n^2 I\right)^{-1}$ in Equation 2.2.

2.2.4 Relationship between neural networks and Gaussian processes

In an early work Neal [10] discusses the asymptotic behavior of an untrained, fully-connected neural network and its convergence to a Gaussian process as the number of hidden nodes approaches infinity. In his dissertation, Neal delivers a convincing argument that under certain conditions a neural network and a Gaussian process model are equivalent. It also provides a mathematical expression that links the respective activation and covariance functions of the two models.

Equivalence of neural networks and Gaussian processes can be studied by investigating the prior joint distributions of model output values. In the discussion below I follow the notation in [23] and the structural diagram of the single-hidden-layer neural network as shown in Figure 3.1.

First, for a fixed input x the expected value at the i^{th} output node for any integer value N_1 is

$$E[z_i^1] = E\Big[b_i^1 + \sum_{j=1}^{N_1} W_{ij}^1 \mathbf{X}_j^1(x)\Big] = 0,$$

since W_{ij}^1 and $\mathbf{X}_j^1(x)$ are independent, and $E[b_i^1] = E[W_{ij}^1] = 0$. I further assume $E[(\mathbf{X}_j^1(x))^2] < 0$. Then, to explore the case when the width N_1, i.e. the number of nodes in the hidden layer, approaches infinity, one considers the random variable $S_j = W_{ij}^1 \mathbf{X}_j^1(x), k^1(x,y) \coloneqq E[\mathbf{X}_j^1(x)\mathbf{X}_j^1(y)]$, and

$$E[S_j] = 0,\ \mathrm{var}[S_j] = \mathrm{var}[W_{ij}^1 \mathbf{X}_j^1(x)] = E[(W_{ij}^1)^2]E[(\mathbf{X}_j^1(x))^2] = \frac{\sigma_w^2}{N_1} k^1(x,x) < 0.$$

Denote $\bar{S}_{N_1} = \frac{1}{N_1}\sum_{j=1}^{N_1} W_{ij}^1 \mathbf{X}_j^1(x)$ and apply the central limit theorem. As $N_1 \to \infty$,

$$\begin{aligned}
&\sqrt{N_1}\Big(\bar{S}_{N_1} - 0\Big) \xrightarrow{\mathrm{D}} N\Big(0, \frac{\sigma_w^2}{N_1} k^1(x,x)\Big) \\
\implies &\sqrt{N_1}\Big(\frac{1}{N_1}\sum_{j=1}^{N_1} W_{ij}^1 \mathbf{X}_j^1(x)\Big) \xrightarrow{\mathrm{D}} N\Big(0, \frac{\sigma_w^2}{N_1} k^1(x,x)\Big) \\
\implies &\sum_{j=1}^{N_1} W_{ij}^1 \mathbf{X}_j^1(x) \xrightarrow{\mathrm{D}} N\Big(0, \sigma_w^2 k^1(x,x)\Big) \\
\implies &z_i^1(x) \xrightarrow{\mathrm{D}} N\Big(0, \sigma_b^2 + \sigma_w^2 k^1(x,x)\Big)
\end{aligned}$$

Therefore, for any finite set of input values $\mathbf{x} = \{x^{(1)}, \cdots, x^{(n)}\}$, the joint probability distribution at the i^{th} output node converges to a multivariate Gaussian distribution with zero mean and covariance:

$$\begin{aligned}
c^1(x,y) &\coloneqq \mathrm{Cov}(z_i^1(x) z_i^1(y)) \\
&= E[z_i^1(x) z_i^1(y)] \\
&= \sigma_b^2 + \sum_{j=1}^{N_1} \frac{\sigma_w^2}{N_1} E[\mathbf{X}_j^1(x)\mathbf{X}_j^1(y)] \\
&= \sigma_b^2 + \sigma_w^2 k^1(x,y).
\end{aligned}$$

Following the analysis in [10], this result gives us a Gaussian process $z_i^1(\mathbf{x}) \sim \mathcal{GP}\big(0, \mathcal{C}^1\big)$, with $\mathcal{C}^1$ being the covariance matrix composed of $c^1(x,y)$, with $x, y \in \mathbf{x}$. This concludes that in the limit of

an infinite number of hidden nodes, an untrained fully-connected, single-hidden-layer feedforward neural network is equivalent to a Gaussian process.

2.3 Conclusion

The notion of NN-GP model equivalence for deep networks has been investigated in recent research. In [23] the authors discuss and derive the equivalence of fully-connected deep neural networks with infinitely wide hidden layers and Gaussian processes. They also propose an efficient computational algorithm to compute Gaussian process covariance functions for large datasets. The work in [38] shows that NN-GP model equivalence can be extended to convolutional neural networks. Specifically, the authors prove that the output of a convolutional neural network with an infinite number of convolutional filters can be represented by a Gaussian process.

The theoretical and empirical findings resulting from these research efforts provide a strong motivation for my investigating Gaussian process models as a statistical learning tool for improving neural network initialization and activation function selection.

Chapter 3

Single-hidden-layer Neural Networks

The overarching question for this chapter is whether neural network initialization can be improved with results from Gaussian process optimization. Here I focus on single-hidden-layer, fully-connected feedforward neural networks with rectified linear unit (ReLU) activation function. First, I explore the architecture of a typical neural network with an input layer, an output layer, and a single hidden layer where each processing node is connected to every node in consecutive layers. Then I examine the relationship between the network activation function and its corresponding Gaussian process covariance function. I present an alternative derivation of the closed-form covariance function at the output of the network model. Next, I describe experiments conducted on simulated data and a real-world dataset for regression prediction. These experiments are designed to measure the efficacy of the proposed GP-initialization method for the selecting initial hyperparameters for network training, Finally, I discuss the experimental results and their implications.

3.1 Notation and Structure of a Single-hidden-layer Neural Network

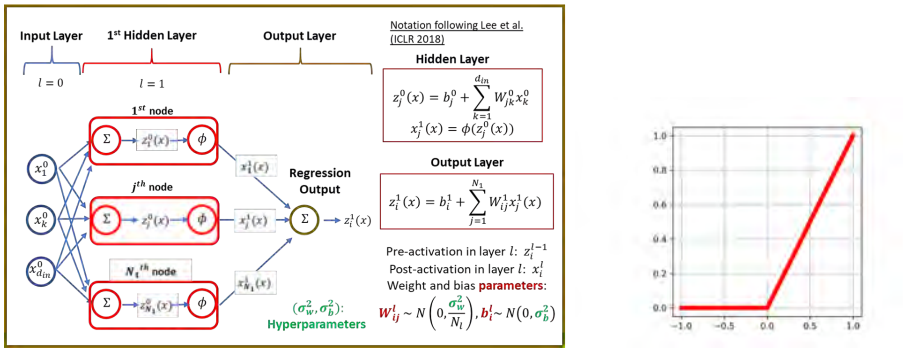

Figure 3.1: Structural diagram of a single-hidden-layer, fully-connected feedforward neural network for regression prediction with a plot of the standarad ReLU activation function.

The left panel in Figure 3.1 shows a single-hidden-layer, fully-connected feedforward neural network for regression prediction. On the right panel is the plot of the standard rectified linear unit (ReLU) activation function. The nonlinearity ϕ outputs the nonnegative part of its input: $\phi(a) \coloneqq (a)_+ = \max(0, a) = a$ for $a \geq 0; \phi(a) = 0$ otherwise.

To maintain consistency in this dissertation I continue to adopt the notation used in [23].

As shown in the structural diagram, the single-hidden-layer neural network has a set of inputs $x = \{x_k^0\}$, $k \in \{1, 2, \cdots, d_{in}\}$ with input layer width $d_{in} \coloneqq N_0$. The notation N_l is used to indicate the width of, or the number of processing nodes in, the l^{th} layer. Let W_{jk}^0 and b_j^0 denote the weight and bias of the connection from k^{th} input node to j^{th} hidden node. I assume that the set of weights and the set of biases are each independent and identically distributed, and that they are mutually independent: $W_{jk}^0 \overset{\text{iid}}{\sim} \mathcal{N}(0, \frac{\sigma_w^2}{N_0}), b_j^0 \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_b^2),$ and $W_{jk}^0 \perp\!\!\!\perp b_j^0$.

Similarly, the weight and the bias parameters from j^{th} hidden node to i^{th} output node with hidden layer width N_1 are given by $W_{ij}^1 \overset{\text{iid}}{\sim} \mathcal{N}(0, \frac{\sigma_w^2}{N_1}), b_i^1 \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_b^2),$ and $W_{ij}^1 \perp\!\!\!\perp b_i^1$. For regression models the output layer has a single node, and therefore $i \in \{1\}$.

The ReLU nonlinearity depicted in the right panel of Figure 3.1 produces only the positive part of its input. It has a slope of 1 for positive inputs. In practice the slope is typically set to 0 for any input ≤ 0.

Each hidden node has two functions: computing the weighted sum of received values to produce the pre-activation, and passing the pre-activation through a nonlinear function to generate the post-activation. The output layer collects post-activations from the hidden layer to give the network output. Here the pre-activation is represented by $z_j^0(x) = b_j^0 + \sum_{k=1}^{d_{in}} W_{jk}^0 x_k^0$, and the post-activation by $x_j^1(x) = \phi(z_j^0(x))$, $j \in \{1, 2, \cdots, N_1\}$. Since one typically applies the linear activation function in the output stage of a regression model, the model output is simply $z_1^1(x) = b_1^1 + \sum_{j=1}^{N_1} W_{1j}^1 x_j^1(x)$.

3.2 ReLU Activation and Covariance Functions

Recent developments in nonlinear activation functions have improved prediction performance significantly in neural networks [4]. Specifically, rectified linear unit (ReLU) provides faster convergence in deep model training in comparison to standard sigmoidal functions.

ReLU is termed the positive part rectifying nonlinearity in [39] and is shown to work well for object recognition. The observation is supported by other studies such as [40] where ReLU in Restricted Boltzmann Machines is shown to outperform binary stochastic units in object recognition and face verification tasks. Furthermore, Glorot et al. [17] demonstrate that rectified activation functions deliver better solutions than hyperbolic tangent function in image recognition and sentiment analysis tasks. Subsequently, ReLU is shown to improve performance in neural network image classifiers [41], acoustic models [42], and neural network speech processing [43].

As discussed in Chapter 2, equivalent neural networks and Gaussian process models are linked through their activation and covariance functions. The next section describes the procedure for obtaining a closed-form representation of the covariance function corresponding to the ReLU activation function.

3.2.1 Derivation of closed-form ReLU covariance function

The ReLU covariance function is developed to estimate model covariance at the output of ReLU neural networks. The derivation given below was inspired by the work on arc-cosine family of kernels developed in [11] which, however, excludes the bias term. In his further investigation, [44, §3.2.2], Cho states that arc-cosine kernels with the inclusion of bias could no longer be expressed in closed-form, but rather in terms of definite integrals. Nevertheless, in this section I show the procedure for obtaining a closed-form expression for ReLU (which is a member of the arc-cosine family) covariance function including both the weight and bias terms from scratch. The resulting expression provides the foundation for building output covariance function of multilayer ReLU

neural networks described in Chapter 4 (an alternative to the expression given in [23]).

I first derive the expectation of the product of post-activations, instead of the expected value of the input to the nonlinearity [23]. Then, I apply the output layer activation function on the post-activation expected value.

I maintain that given the input (x, y) the exact expression for the covariance function at the network output is

$$\begin{aligned}\mathcal{C}(x,y) &\coloneqq E[(b_i^1)^2] + \sum_{j=1}^{N_1} E[(W_{ij}^1)^2] E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)] \\ &= \sigma_b^2 + \frac{\sigma_w^2}{2\pi}\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sin\,\phi + (\pi-\phi)\,\cos\,\phi\Big), \\ \text{where } \phi &= \cos^{-1}\left\{\frac{\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x\cdot y)\big)}{\Big((\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)^{1/2}}\right\}. \qquad \square\end{aligned}$$

In this section I briefly describe the steps taken to arrive at the expression. The complete derivation is provided in Appendix A.1.

Referring to Figure 3.1, I consider input vectors $x^0, y^0 \in \mathcal{R}^{d_{in}}$. The initial weight value is drawn randomly from the Gaussian distribution $f_{W_{jk}^0} = \mathcal{N}(0, \frac{\sigma_w^2}{d_{in}})$ and bias value from $f_{b_j^0} = \mathcal{N}(0, \sigma_b^2)$. The expected value of the product of post-activations at the output of the j^{th} hidden node is computed as

$$\begin{aligned}&E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)] \\ &= \int\cdots\int_{-\infty}^{\infty} \max(b_j^0 + w_j^0\cdot x^0)\,\max(b_j^0 + w_j^0\cdot y^0) f_{b_j^0, W_j^0}(b,w)\,dw_j^0\,db_j^0 \\ &= \int\cdots\int_{-\infty}^{\infty} (b_j^0 + w_j^0\cdot x^0)_+ (b_j^0 + w_j^0\cdot y^0)_+ f_{b_j^0, W_j^0}(b,w)\,dw_j^0\,db \qquad (3.1)\end{aligned}$$

The pre-activations can be written in terms of random variables U, V as

$$\begin{aligned}U = b_j^0 + W_j^0\cdot x^0 = b_j^0 + \sum_{k=1}^{d_{in}} W_{jk}^0 x_k^0 \sim \mathcal{N}(0, \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2), \\ V = b_j^0 + W_j^0\cdot y^0 = b_j^0 + \sum_{k'=1}^{d_{in}} W_{jk'}^0 y_{k'}^0 \sim \mathcal{N}(0, \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2).\end{aligned}$$

It can be shown that the random variables U, V have a joint Gaussian distribution:

$$\begin{pmatrix} U \\ V \end{pmatrix} \sim \mathcal{N}(0, \Sigma), \text{ where } \Sigma = \begin{pmatrix} \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2 & \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y) \\ \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y) & \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2 \end{pmatrix},$$

For simplicity I let $x = x^0, y = y^0$. I can therefore write expression (3.1) as

$$\iint_0^\infty uv \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} \exp\big(-\frac{1}{2}(u,v)\Sigma^{-1}(u,v)^T\big)\, du\, dv.$$

Now let $D \coloneqq |\Sigma| = \big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2\big)\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2\big) - \big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)^2$, and

$$\Sigma^{-1} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \text{ where } a_{11} = \frac{1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2),\ a_{22} = \frac{1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2),$$
$$a_{12} = a_{21} = \frac{-1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)).$$

With polar coordinate transformation: $u = \frac{r}{\sqrt{a_{11}}}\cos\alpha$, $v = \frac{r}{\sqrt{a_{22}}}\sin\alpha$, expression (3.1) can be further reduced to

$$\frac{1}{4\pi\mathcal{D}^{1/2}a_{11}a_{22}} \int_{\alpha=0}^{\frac{\pi}{2}} \frac{2\sin 2\alpha}{\Big(1 - \cos\phi \sin 2\alpha\Big)^2}\, d\alpha$$
$$= \frac{1}{2\pi\mathcal{D}^{1/2}a_{11}a_{22}\sin^3\phi}\Big(\sin(\phi) + (\pi - \phi)\cos(\phi)\Big), \text{ where } \phi = \cos^{-1}\Big(\frac{-a_{12}}{\sqrt{a_{11}a_{22}}}\Big).$$

With some algebraic operations and after computing the entries in Σ^{-1}, we arrive at

$$E[\mathbf{X}_j(x)\mathbf{X}_j(y)]$$
$$= \frac{1}{2\pi}\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sin\phi + (\pi - \phi)\cos\phi\Big)$$
$$\text{where } \phi = \cos^{-1}\left\{\frac{\sigma_b^2 + (x \cdot y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}\right\}.$$

The covariance function at the network output is therefore determined to be

$$E\Big[\big(b_i^1 + \sum_{j=1}^{N_1} W_{ij}^1\mathbf{X}_j(x)\big)\big(b_i^1 + \sum_{k=1}^{N_1} W_{ik}^1\mathbf{X}_k(y)\big)\Big] - E\Big[b_i^1 + \sum_{j=1}^{N_1} W_{ij}^1\mathbf{X}_j(x)\Big]\Big[b_i^1 + \sum_{k=1}^{N_1} W_{ik}^1\mathbf{X}_k(y)\Big]$$
$$= E[(b_i^1)^2] + \sum_{j=1}^{N_1} E[(W_{ij}^1)^2]E[\mathbf{X}_j(x)\mathbf{X}_j(y)]$$

$$= \sigma_b^2 + \frac{\sigma_w^2}{N_1} N_1 E[\mathbf{X}_j(x)\mathbf{X}_j(y)]$$
$$= \sigma_b^2 + \frac{\sigma_w^2}{2\pi}\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sin\,\phi + (\pi-\phi)\,\cos\,\phi\Big). \qquad \blacksquare$$

It is important to note that the expression, at a given fixed input, is a function solely dependent on the hyperparameter pair (σ_w^2, σ_b^2). This simplicity enables us to leverage Gaussian process optimization to systematically learn just two values from data, making model tuning a lot more efficient.

3.3 Evaluation of GP-initialization Method

To study the effect of GP-init approach on single-hidden-layer ReLU network model prediction, I conduct a neural network regression experiment on simulated data to observe the change in the network behavior. This is then followed by a similar experiment on the MNIST image dataset. In each experiment network prediction accuracy based on GP-init is computed and compared to the benchmark based on He-initialization method [4]. He-init is a standard weight initialization method developed for ReLU networks. In its formulation, He-init sets the bias term to 0 while the weight values at the l^{th} hidden layer are drawn randomly from a Gaussian distribution with zero mean and a standard deviation equal to $\sqrt{\frac{2}{n_l}}$, where n_l denotes the layer width. He-init is used frequently in deep learning. For that reason it is chosen as the performance benchmark.

Both simulated and MNIST experiments share the same three-phase workflow process as depicted in Figure 3.2. In the first phase, data are randomly divided into training and test sets. Through Gaussian process log marginal likelihood maximization, GP-init method learns hyperparameter values (σ_w^2, σ_b^2) directly from training data. During phase two, a ReLU neural network model is initialized prior to training with the hyperparameter pair obtained based on GP-init. Concurrently, a second network model with identical structure is initialized with hyperparameter pair (2.0, 0.0) in accordance with He-init. The two models are then trained with the same training dataset. In the last phase both trained models are evaluated on the same test set. Model prediction accuracies

based on the two initialization methods are then computed and compared.

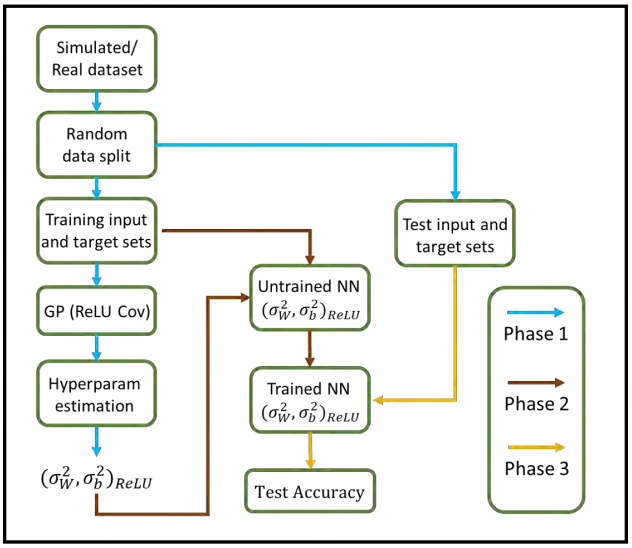

Figure 3.2: Workflow diagram for validating GP-init method in single-hidden-layer ReLU network prediction.

3.3.1 Regression simulation

The main purpose of performing simulations is to explore and understand some properties of the models I aim to study. This is possible because simulations provide a controlled environment where the model structures are customized and the test data are prepared with specific parametric values. Simulation results can therefore offer insight into the model behavior.

The first step in Figure 3.2 includes generating simulated datasets and estimating the hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{ReLU}$.

Generating simulated Data

I use Gaussian process models with ReLU covariance function to generate one-dimensional samples for simulation experiments. The design hyperparameter pairs (σ_w^2, σ_b^2) are chosen with $\sigma_w^2 \in [0.5, 2.0, 10.0]$ and $\sigma_b^2 \in [0.0, 1.0]$. These values are so chosen because I am interested in performance comparison at the He-init specification ($\sigma_w^2 = 2$) and comparisons farther away ($\sigma_w^2 = 0.5, 10.0$)

from He-init specification.

For each hyperparameter pair a set of 100 data points of {location, target} are produced and divided randomly into 70 training and 30 test points, where location $\in [0.0, 0.01, \cdots, 1.0]$. All together, 6 datasets are generated. These datasets are associated with the six design hyperparameter pairs on which the GP-init method is assessed. They allow us to train Gaussian process models (3x2 = 6 total, one for each hyperparamter pair) to estimate hyperparameters, calculate log marginal likelihood based on GP-init and He-init methods, measure and compare prediction error of corresponding neural network models.

Estimating hyperparameters from data

Given a simulated dataset, I first estimate its design hyperparameter pair via maximizing ReLU Gaussian process log marginal likelihood using Equation 2.3. Then I apply the grid-searching method that iterates through every hyperparameter pair in a specific grid and compute its log marginal likelihood. The pair that attains the maximum log marginal likelihood over the grid is the desired hyperparameter pair for that particular dataset.

At each design weight hyperparameter value I focus on its close neighbors to create the search grid. For $\sigma_w^2 = 0.5$, the grid consists of 21 locations evenly distributed between 0.3 and 0.7. The search grid for $\sigma_w^2 = 2.0$ contains 21 locations between 1.5 and 2.5. Lastly, the search grid for $\sigma_w^2 = 10.0$ also contains 21 locations, evenly spaced between 9.0 and 11.0. The search grid for design bias hyperparameter value is composed of 31 locations ranging from 0.0 to 1.5 with step size set to 0.05. Table 3.1 shows the Ground Truth design pairs (σ_w^2, σ_b^2), the hyperparameter pairs learned from training data, and the associated log marginal likelihood values (larger values are better) based on GP-init and He-init methods, respectively.

The estimates from GP-init for the six datasets are used in the next section to evaluate the usefulness of GP-init for neural network initialization.

Table 3.1: Log marginal likelihood (lml) values for single-layer model computed based on GP-init and He-init methods.

Ground Truth	GP-init		He-init	
	(σ_w^2, σ_b^2)	lml	(σ_w^2, σ_b^2)	lml
(0.5, 0.0)	(0.3, 0.0)	-26.26	(2.0, 0.0)	-27.80
(0.5, 1.0)	(0.3, 1.5)	-29.29	(2.0, 0.0)	-202.68
(2.0, 0.0)	(2.5, 0.0)	-31.19	(2.0, 0.0)	-31.43
(2.0, 1.0)	(1.5, 1.5)	-23.26	(2.0, 0.0)	-456.27
(10.0, 0.0)	(9.0, 0.0)	-25.90	(2.0, 0.0)	-27.00
(10.0, 1.0)	(11.0, 0.2)	-39.94	(2.0, 0.0)	-160.97

Regression implementation details

All neural network models for the simulation experiments share the same feedforward, fully-connected, single-hidden-layer architecture. The only difference among the models is the hyperparameter pair for initialization. Since all inputs are one-dimensional, a hidden-layer width of 3 is sufficient for training and testing. For model compilation, Adam optimizer is selected along with 'mse' loss function and 'mse' evaluation metric. Model training is then performed with a batch size of 24 examples, and 200 passes (epochs) of the complete training set. During testing, each trained model is evaluated repeatedly to collect 50 MSE values whose median is reported as the test result for performance comparison.

3.3.2 Assessing GP-init on neural network regression performance

The second step of the simulation experiment is training neural network models. For each training dataset, a neural network model is initialized with the estimated hyperparameter pair corresponding to the dataset. For example, for the dataset generated with Ground Truth $(\sigma_w^2, \sigma_b^2) = (0.5, 1.0)$, the neural network model is initialized with the GP-init recommended pair (0.3, 1.5) (see Table

3.1). At the same time, an identically structured neural network model is initialized with (2.0, 0.0) according to the He-init method. Both GP-init and He-init neural networks are then trained over 200 epochs.

The final step is evaluating and comparing regression performance of GP-init against He-init benchmark. This is done by performing regression with the trained neural network models on test datasets.

The results are shown in Table 3.2. The table lists the Ground Truth design pairs (σ_w^2, σ_b^2) and the hyperparameter pairs learned as described in the previous section. The table also contains the associated neural network prediction errors (MSEs) (smaller values are better) based on GP-init and He-init methods, respectively.

Table 3.2: Neural network prediction error (MSE) values for single-layer model computed based on hyperparameter pair (σ_w^2, σ_b^2) obtained from respectively GP-init and He-init methods.

Ground Truth	GP-init		He-init		$\frac{\text{GP-init MSE}}{\text{He-init MSE}}$
	(σ_w^2, σ_b^2)	MSE	(σ_w^2, σ_b^2)	MSE	Ratio
(0.5, 0.0)	(0.3, 0.0)	0.1039	(2.0, 0.0)	0.1067	0.97
(0.5, 1.0)	(0.3, 1.5)	0.0888	(2.0, 0.0)	0.2045	0.43
(2.0, 0.0)	(2.5, 0.0)	0.2736	(2.0, 0.0)	0.2945	0.93
(2.0, 1.0)	(1.5, 1.5)	0.3849	(2.0, 0.0)	0.4156	0.93
(10.0, 0.0)	(9.0, 0.0)	0.3267	(2.0, 0.0)	0.2766	1.18
(10.0, 1.0)	(11.0, 0.2)	0.8683	(2.0, 0.0)	1.0716	0.81

The relative performance between GP-init and He-init is measured by their prediction error ratio, $\frac{\text{GP-init MSE}}{\text{He-init MSE}}$ and is illustrated in Figure 3.3. Since the He-init prediction error is used as the reference, a value less than 1 in the error ratio signifies the better performance of the GP-init method,

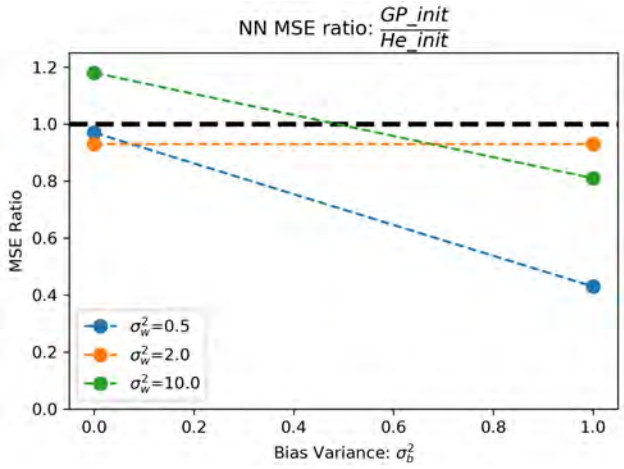

Figure 3.3: Performance ratio in neural network regression prediction, $\frac{\text{GP-init MSE}}{\text{He-init MSE}}$, from the single-hidden-layer neural network regression experiment.

Discussion

We make a number of observations about the experimental results

- GP-init achieves better regression accuracy than He-init in the majority of cases
- GP-init works better than He-init for inputs characterized by non-zero σ_b^2 value
- GP-init leads to lower regression error for inputs characterized by smaller σ_w^2 values
- GP-init works slightly better than He-init for inputs characterized by $\sigma_w^2 = 2.0$

The experiments suggest that both hyperparameter values, σ_w^2 and σ_b^2, are key components in the formulation of the model output covariance function and the bias term should not be left out. Furthermore, it indicates that GP-init approach may indeed utilize information extracted from data to achieve better network initialization than He-init method.

3.3.3 Image prediction

Experiment objective

Our goal here is to assess GP-init method for recommending hyperparameter initialization in a neural network image prediction task.

As in the simulation experiments, I start with splitting the input, MNIST image database in this case, randomly into training and test datasets. Recalling that the MNIST database [22] is a standard image prediction database consisting of a training set of 60,000 examples, and a test set of 10,000 examples of handwritten digits. Each example is a 28 x 28 gray-level image. Some examples are depicted below in Figure 3.4.

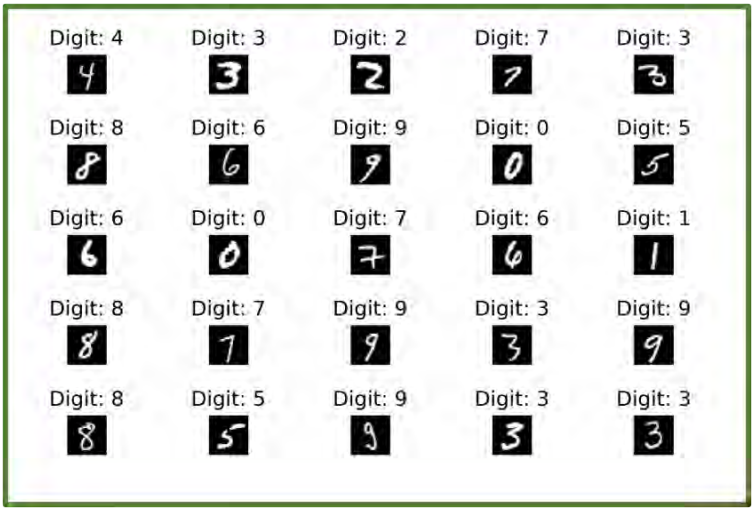

Figure 3.4: Examples of MNIST handwritten digit images.

While the entire test set is retained for evaluation, nine subsets are randomly selected from the complete training dataset, varying from 1000 to 9000 training examples. The reason is two-fold: (1) to examine consistency in GP-init recommendation across various training sizes, and (2) to conduct experiments within the limitations of available hardware capacity.

Estimating hyperparameters from data

Similar to the procedure applied in the simulation experiments, the hyperparameter pair (σ_w^2, σ_b^2) for each training dataset is learned through the maximization of the ReLU Gaussian process log marginal likelihood. However, contrary to the simulation experiments, the hyperparameter pair based on which MNIST data was generated is unknown. I am therefore interested in exploring the behavior of the models over the ranges $(\sigma_w^2)_{ReLU} \in [0.4, 1.2, 2.0, 2.8, 3.6]$ and $(\sigma_b^2)_{ReLU} \in [0, 1.0, 2.0]$. These ranges are chosen after reviewing the experimental values used in [20, 21].

The experiments conducted on the MNIST dataset show that the log marginal likelihood maximization procedure determined consistently the pair (3.6, 0.0) as the optimal hyperparameter pair for all training sizes from 10000 to 9000 examples.

Prediction implementation details

As for the simulation experiments, all neural network models for the prediction experiments share the same feedforward, fully-connected, single-hidden-layer architecture. The main difference among the models is the hyperparameter pair for initialization. Since the input dimension is 28 x 28 = 784 (pixels), each hidden-layer is assigned 2000 nodes for training and testing. For model compilation, Adam optimizer is selected along with 'mse' loss function and 'accuracy' evaluation metric. Model training is then performed with a batch size of 64, and 250 passes of the complete training set.

Assessing GP-init on neural network prediction performance

Following the workflow diagram shown in Figure 3.2, for each training dataset generated with a specific training size, a neural network model is initialized based on the GP-init method. It is initialized prior to training with the estimated hyperparameter pair obtained through log marginal likelihood. A separate neural network model is initialized with $(\sigma_w^2, \sigma_b^2) = (2.0, 0.0)$ before training in accordance with the He-init method.

Table 3.3: GP-init vs. He-init: comparing prediction accuracy on MNIST image dataset for a single-hidden-layer network model.

Training Size	Best Case		Worst Case		He-Init		GP-init	
	Acc.	(σ_w^2, σ_b^2)	Acc.	(σ_w^2, σ_b^2)	Acc.	(σ_w^2, σ_b^2)	Acc.	(σ_w^2, σ_b^2)
1000	92.68	(0.4, 0)	88.18	(3.6, 2)	**92.12**	**(2, 0)**	90.70	(3.6, 0)
2000	94.69	(3.6, 0)	92.99	(3.6, 2)	94.22	(2, 0)	**94.69**	**(3.6, 0)**
3000	95.07	(0.4, 0)	94.02	(0.4, 2)	94.59	(2, 0)	**94.82**	**(3.6, 0)**
4000	95.22	(0.4, 0)	93.97	(0.4, 2)	94.70	(2, 0)	**94.97**	**(3.6, 0)**
5000	95.53	(0.4, 0)	94.31	(0.4, 2)	95.33	(2, 0)	**95.41**	**(3.6, 0)**
6000	95.94	(1.2, 0)	95.22	(1.2, 2)	**95.76**	**(2, 0)**	95.73	(3.6, 0)
7000	96.24	(0.4, 0)	95.29	(0.4, 2)	96.09	(2, 0)	**96.12**	**(3.6, 0)**
8000	96.40	(0.4, 0)	95.54	(1.2, 2)	95.89	(2, 0)	**95.96**	**(3.6, 0)**
9000	96.64	(0.4, 0)	95.74	(2, 2)	96.37	(2, 0)	**96.42**	**(3.6, 0)**

The final step is measuring and comparing prediction performance of GP-init against He-init benchmark on the test dataset.

Table 3.3 presents the prediction results, comparing GP-init to He-init, the best and the worst performer for various training sizes. The same results are also shown in Figure 3.5 for visual comparison, with the x-axis representing the training size and the y-axis prediction accuracy.

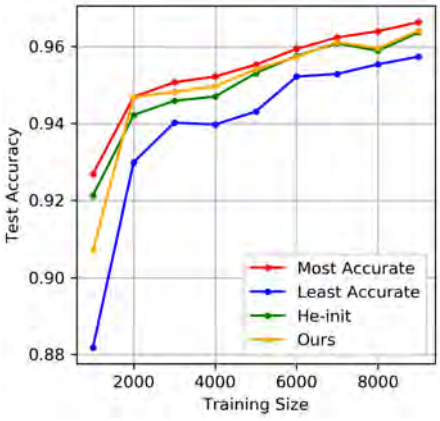

Figure 3.5: Comparison of GP-init to He-init in prediction accuracy on MNIST image dataset for a single-hidden-layer network model.

Discussion

From the experiments we observe that GP-init achieves slightly higher prediction accuracy than He-init in most cases. It generates consistent estimates of the hyperparameter pair (σ_w^2, σ_b^2). Lastly, GP-init learns that the optimal σ_b^2 ought to be set to 0 for the MNIST dataset.

3.4 Conclusion

The main research question to address in this chapter is if GP-initialization method based on Gaussian process optimization can improve training in single-hidden-layer neural networks, thereby improving their prediction accuracy.

In order to conduct experiments for testing the proposed hypothesis, I first obtain an alternative derivation of the model output covariance function shared by corresponding neural network and Gaussian process models. Next, I apply log marginal likelihood maximization on input data to learn the hyperparameter pair associated with the dataset. Neural network models are then initialized with the estimates prior to training. Finally, the trained networks are evaluated over test datasets. The procedure is carried out for a simulated and a real-world regression task.

The simulation experiments suggest that using the model output covariance function completed with both weight and bias components, GP-init approach learns information from data and improves network initialization. In addition, results from the image prediction experiment suggest that GP-init achieves prediction accuracy comparable to, if not better than, He-initialization method. GP-init method also generates consistent hyperparameter estimates over various training sizes. This finding is important because the likelihood optimization procedure performs hyperparameter estimation significantly faster with a much smaller training set, consequently reducing computational cost.

In short, these empirical results show that the initialization scheme based on log marginal likelihood maximization is a promising approach to improving model initialization in fully-connected

single-hidden-layer neural networks. However, additional experiments with a larger design space of hyperparameter pairs (σ_w^2, σ_b^2) will be valuable to ascertaining current findings. It will also be beneficial to conduct simulations with multi-dimensional inputs and observe changes in model behavior from one-dimensional inputs.

Chapter 4

Multi-hidden-layer Neural Networks

Motivated by the connection between deep neural networks and Gaussian processes [45] [23] [38], I investigate in this chapter the benefits of applying GP-initialization for neural networks with 2- and 3-hidden layers, respectively. Following the procedures established in Chapter 3, for each multilayer model I aim to identify the hyperparameter pair (σ_w^2, σ_b^2) that maximizes the model log marginal likelihood (lml). Then, evaluation of the performance ratio in regression prediction, $\frac{\text{GP-init MSE}}{\text{He-init MSE}}$, provides information on the usefulness of GP-init method.

First, deriving the recursive covariance function is essential for evaluating GP-init for deep structure initialization. The reason is that a multilayer neural network is also a prediction model with an output covariance that can be expressed in terms of the model initial hyperparameter values and its activation function. Therefore a multilayer neural network ought to have a corresponding deep Gaussian process with approximately the same output covariance.

The derived recursive covariance function will enable applying log marginal likelihood maximization to measure the effectiveness of recommendations made by GP-init method in multilayer models. In the next section, I examine the structure of a two-hidden-layer fully-connected neural network to help extending the derivation of model output covariance function from single- to multilayer networks.

As in Section 3.3, I conduct regression experiments on simulated data and the MNIST image dataset to validate the efficacy of GP-init on multilayer neural networks.

4.1 Structure of a Multi-hidden-layer Neural Network

Similar to the single-hidden-layer model described in Chapter 3, the multilayer neural network as shown in Figure 4.1 has inputs $x = \{x_k^0\},\ k \in \{1, 2, \cdots, d_{in}\}$ where d_{in} denotes the input

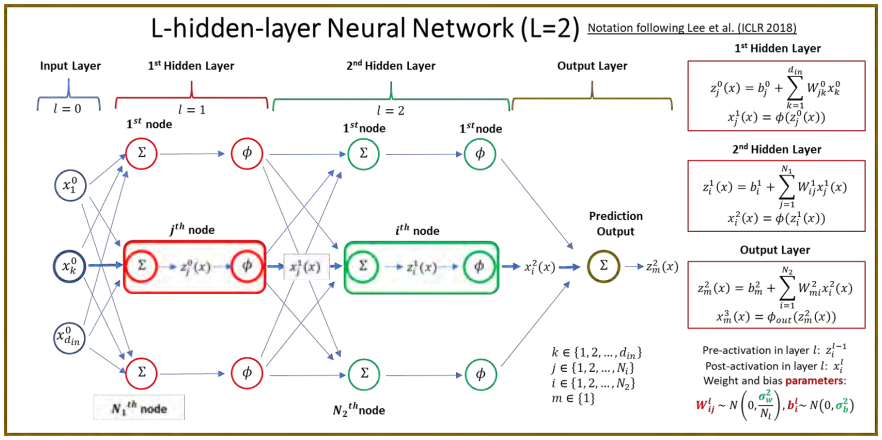

Figure 4.1: Configuration diagram of a fully-connected, two-hidden-layer feedforward neural network for regression prediction.

dimension. I again use the notation N_l to indicate the width of, or the number of processing nodes in, the l^{th} layer. So $d_{in} = N_0$. Let W_{jk}^0 and b_j^0 denote the weight and bias of the connection from k^{th} input node to j^{th} hidden node. In addition, let W_{ij}^1 and b_i^1 denote the weight and bias of the connection from j^{th} hidden node in the first hidden layer to i^{th} hidden node in the second hidden layer. I assume that the set of weights and the set of biases are each independent and identically distributed, and that they are mutually independent, i.e., $W_{jk}^0 \overset{\text{iid}}{\sim} \mathcal{N}(0, \frac{\sigma_w^2}{N_0}), b_j^0 \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_b^2)$, and $W_{jk}^0 \perp\!\!\!\perp b_j^0$; $W_{ij}^1 \overset{\text{iid}}{\sim} \mathcal{N}(0, \frac{\sigma_w^2}{N_1}), b_i^1 \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_b^2)$, and $W_{ij}^1 \perp\!\!\!\perp b_i^1$. Moreover, $W_{mi}^2 \overset{\text{iid}}{\sim} \mathcal{N}(0, \frac{\sigma_w^2}{N_2}), b_m^2 \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_b^2)$, and $W_{mi}^2 \perp\!\!\!\perp b_m^2$. For regression models the output layer has a single node, and therefore $m \in \{1\}$. The pre-activation and post-activation for the second hidden layer are given by, respectively, $z_i^1(x) = b_i^1 + \sum_{j=1}^{N_1} W_{ij}^1 x_j^1$ and $x_i^2(x) = \phi(z_i^1(x))$, $i \in \{1, 2, \cdots, N_2\}$. As a result, the model output is $z_1^2(x) = b_1^2 + \sum_{i=1}^{N_2} W_{1i}^2 x_i^2(x)$.

4.2 Derivation of Closed-form Recursive ReLU Covariance Function

The recursive ReLU covariance function for a multilayer prediction model can be obtained based on the closed-form expression of the output covariance function for a single-hidden-layer ReLU neural network established in Chapter 3. Importantly, it is known that for any covariance function k, which

is inherently symmetric and positive semidefinite, there exists a feature map Φ and an inner product space $F \coloneqq \text{span}\{\Phi(x) : x \in \mathcal{X}\}$ with an inner product $\langle \cdot, \cdot \rangle$ such that $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$.

Following the reasoning in [11, §2.3], I apply $l-$layer nested feature maps evaluated at the inputs $x, y \in \mathcal{R}^{d_{in}}$ to construct the recursive covariance function corresponding to ReLU neural network with l hidden layers:

$$k^{(l)}(x, y) = \langle \underbrace{\Phi(\Phi(\cdots \Phi(x)) \cdots)}_{l \text{ terms}}, \underbrace{\Phi(\Phi(\cdots \Phi(y)) \cdots)}_{l \text{ terms}} \rangle.$$

Given inputs $x, y \in \mathcal{R}^{d_{in}}$, I claim that for any whole number l the exact recursive ReLU covariance function of the l^{th}-layer post-activation is determined to be

$$k^{(l)}(x, y) = \frac{1}{2}\Big(\sigma_b^2 + k^{(l-1)}(x, x)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + k^{(l-1)}(y, y)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\xi^{(l)}(x, y) \tag{4.1}$$

$$\text{with } \xi^{(l)}(x, y) = \frac{1}{\pi}\Big(\sin\,\theta^{(l)}(x, y) + (\pi - \theta^{(l)}(x, y))\,\cos\,\theta^{(l)}(x, y)\Big),$$

$$\text{and } \theta^{(l)}(x, y) = \cos^{-1}\left\{\frac{\sigma_b^2 + k^{(l-1)}(x, y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + k^{(l-1)}(x, x)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + k^{(l-1)}(y, y)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}\right\}.$$

I define $k^{(0)}(x, y) \coloneqq \langle x, y \rangle$ at the input layer.

As a result, given input (x, y) the effective covariance function at the output node p for an $L-$hidden-layer ReLU neural network is simply:

$$E[(b_p^L)^2] + \sum_{q=1}^{N_L} E[(W_{pq}^L)^2]E[\mathbf{X}_q(x)\mathbf{X}_q(y)]$$

$$= \sigma_b^2 + \sigma_w^2 k^{(L)}(x, y). \tag{4.2}$$

The complete derivation is provided in Appendix A.2.

4.3 Evaluation of GP-init Method

This section studies the effect of GP-init approach on multi-hidden-layer ReLU networks. Experiments are conducted to measure predictive performance of multilayer neural networks on simulated

data and the MNIST image dataset for regression tasks. The overall objective is to assess whether GP-init is a better choice than He-init method for multilayer-network prediction.

As in Chapter 3, these experiments are conducted following the three-phase workflow process shown in Figure 3.2. First, the data are partitioned randomly into training and test sets. Then, GP-init learns hyperparameter values (σ_w^2, σ_b^2) from training data via log marginal likelihood maximization. Prior to model training, a network is initialized with GP-init recommended setting. Simultaneously, an identically built neural network is initialized according to the He-init method. The trained models are then evaluated using the same test dataset. Neural network prediction accuracies are then computed and compared.

4.3.1 Regression simulation

The procedures described in this section parallel those in Chapter 3. The main difference is that here I am investigating if GP-init provides useful recommendations for initialization in multilayer instead of single-hidden-layer neural networks. Specifically, the objective is to determine if GP-init could improve neural network prediction accuracy on regression tasks for network depths: $L = 2$ and 3.

Generating simulated data

I first generate one-dimensional datasets using a Gaussian process with the recursive ReLU covariance function expressed in equation (4.2). The design hyperparameter pairs (σ_w^2, σ_b^2) are again chosen with $\sigma_w^2 \in [0.5, 2.0, 10.0]$ and $\sigma_b^2 \in [0.0, 1.0]$, producing six distinct design points.

For each network depth L and each hyperparameter pair, a set of 100 data points of {location, target} is produced and split randomly into 70 training and 30 test points. In all, the process generates 3x2x2 = 12 sets of data.

These datasets are associated with the twelve design hyperparameter pairs on which the GP-init method is assessed. They are used to train the twenty-four corresponding Gaussian process models

for estimating hyperparameters, calculating log marginal likelihood based on GP-init and He-init methods. The experiments are completed with measuring and comparing prediction error of the neural network models.

Estimating hyperparameters from data

Estimation of design hyperparameter pairs is performed via maximizing the Gaussian process log marginal likelihood. I apply the grid-searching method over a specific set of hyperparameter pairs. The pair that attains the maximum log marginal likelihood is the desired hyperparameter pair for that particular dataset. The search-grid for $\sigma_w^2 = 0.5$ contains 21 locations evenly distributed between 0.3 and 0.7. Similarly, the search grid for $\sigma_w^2 = 2.0$ contains 21 locations between 1.5 and 2.5. Also, the search grid for $\sigma_w^2 = 10.0$ consists of 21 locations evenly spaced between 9.0 and 11.0. The search grid for design bias hyperparameter value has 31 locations ranging from 0.0 to 1.5 with step size set to 0.05. The results are shown in Table 4.1.

Table 4.1 presents the Ground Truth design pairs (σ_w^2, σ_b^2), the hyperparameter pairs learned from training data, and the associated log marginal likelihood values based on respectively GP-init and He-init methods. The estimates obtained from GP-init for the twelve datasets are then used to validate the GP-init method for neural network initialization.

Regression implementation details

All twenty-four models for the simulation experiment are feedforward, fully-connected neural networks. Since all inputs are one-dimensional, a hidden-layer width of 3 is sufficient for model training and testing. The other difference among the models is the hyperparameter pair for initialization. For model compilation, Adam optimizer is selected along with 'mse' loss function and 'mse' evaluation metric. Model training is then conducted with a batch size of 24 examples, and 200 passes of the entire training set (epochs).

Table 4.1: Log marginal likelihood (lml) values corresponding to neural networks with 2 and 3 hidden layers computed based on GP-init and He-init methods, respectively.

Ground Truth	GP-init		He-init	
	(σ_w^2, σ_b^2)	lml	(σ_w^2, σ_b^2)	lml
L2: (0.5, 0.0)	(0.52, 0.0)	-66.36	(2.0, 0.0)	-128.86
L2: (0.5, 1.0)	(0.46, 0.9)	-62.78	(2.0, 0.0)	-128.72
L2: (2.0, 0.0)	(2.45, 0.0)	-83.27	(2.0, 0.0)	-86.61
L2: (2.0, 1.0)	(2.40, 0.0)	-83.10	(2.0, 0.0)	-86.21
L2: (10.0, 0.0)	(11.0, 0.5)	-199.53	(2.0, 0.0)	-1478.58
L2: (10.0, 1.0)	(10.7, 0.65)	-192.03	(2.0, 0.0)	-1265.66
L3: (0.5, 0.0)	(0.46, 0.0)	-4.82	(2.0, 0.0)	-117.36
L3: (0.5, 1.0)	(0.48, 0.2)	-9.44	(2.0, 0.0)	-117.38
L3: (2.0, 0.0)	(2.15, 0.1)	-38.85	(2.0, 0.0)	-44.96
L3: (2.0, 1.0)	(2.4, 0.0)	-49.47	(2.0, 0.0)	-56.65
L3: (10.0, 0.0)	(11.0, 0.0)	-229.56	(2.0, 0.0)	-6510.00
L3: (10.0, 1.0)	(11.0, 0.0)	-224.85	(2.0, 0.0)	-6428.89

4.3.2 Assessing GP-init on multilayer neural network regression performance

Next in the simulation experiment is training neural network models. For each dataset generated with Ground Truth hyperparameter pair, a neural network model is initialized prior to training with the estimated hyperparameter pair. For example, for the dataset generated with Ground Truth $(\sigma_w^2, \sigma_b^2) = (2.0, 1.0)$ and $L = 2$, the neural network model for evaluating GP-init is initialized with the GP-init recommended pair (2.4, 0.0) (see Table 4.1). A separate neural network model for evaluating the He-init method is initialized with (2.0, 0.0) before training.

The final step computes and measures regression performance of GP-init against He-init benchmark. This is done by performing regression with the trained neural network models on the test datasets.

Table 4.2: Neural network prediction error (MSE) values corresponding to neural networks with 2 and 3 hidden layers computed based on GP-init and He-init methods, respectively.

Ground Truth	GP-init		He-init		$\frac{\text{GP-init MSE}}{\text{He-init MSE}}$
	(σ_w^2, σ_b^2)	MSE	(σ_w^2, σ_b^2)	MSE	Ratio
L2: (0.5, 0.0)	(0.52, 0.0)	0.2854	(2.0, 0.0)	0.2894	0.99
L2: (0.5, 1.0)	(0.46, 0.9)	0.3821	(2.0, 0.0)	0.4343	0.88
L2: (2.0, 0.0)	(2.45, 0.0)	0.5853	(2.0, 0.0)	0.6014	0.97
L2: (2.0, 1.0)	(2.40, 0.0)	0.4151	(2.0, 0.0)	0.4276	0.97
L2: (10.0, 0.0)	(11.0, 0.5)	17.2579	(2.0, 0.0)	18.2235	0.95
L2: (10.0, 1.0)	(10.7, 0.65)	14.2456	(2.0, 0.0)	13.5472	1.05
L3: (0.5, 0.0)	(0.46, 0.0)	0.0627	(2.0, 0.0)	0.0671	0.93
L3: (0.5, 1.0)	(0.48, 0.2)	0.0846	(2.0, 0.0)	0.1588	0.53
L3: (2.0, 0.0)	(2.15, 0.1)	0.1211	(2.0, 0.0)	0.1199	1.01
L3: (2.0, 1.0)	(2.4, 0.0)	0.2424	(2.0, 0.0)	0.2676	0.91
L3: (10.0, 0.0)	(11.0, 0.0)	21.2186	(2.0, 0.0)	21.1817	1.00
L3: (10.0, 1.0)	(11.0, 0.0)	31.4627	(2.0, 0.0)	31.2506	1.01

The ratio, $\frac{\text{GP-init MSE}}{\text{He-init MSE}}$, is used to assess the relative performance between GP-init and He-init methods.

The results are shown in Table 4.2. The table lists the Ground Truth design pairs (σ_w^2, σ_b^2), the hyperparameter pairs learned in the previous section, and the associated neural network prediction errors (MSEs) (smaller values are better) based on GP-init and He-init methods, respectively. The relative performance between GP-init and He-init methods is illustrated in Figure 4.2.

Discussion

We make the following observations about the experimental results. GP-init in general performs as well as, if not better than, He-init in either two- or three-hidden-layer models. Also, the difference

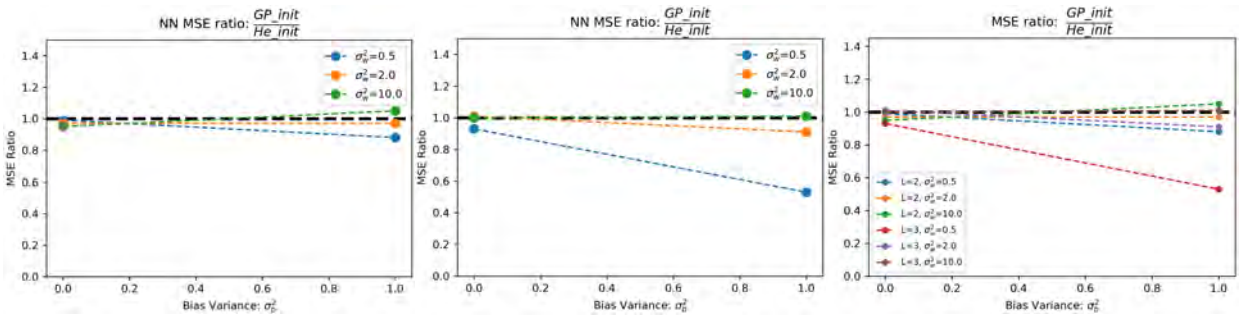

Figure 4.2: Performance ratio in regression prediction, $\dfrac{\text{GP-init MSE}}{\text{He-init MSE}}$, resulting from ReLU neural networks with two and three hidden layers, respectively.

in prediction accuracy between GP-init and He-init is more noticeable at $\sigma_b^2 = 1.0$. Furthermore, the one major difference occurs at $(\sigma_w^2, \sigma_b^2) = (0.5, 1.0)$ where GP-init achieved about 50% of the He-init error rate

The experiments suggest that both hyperparameter values, σ_w^2 and σ_b^2, are important for constructing recursive covariance functions. Moreover, they indicate that GP-init approach may be able to extract useful information from data to enhance initialization in multilayer networks.

4.3.3 Image prediction

The prediction experiments are conducted on the MNIST database [22] which consists of a training set of 60,000 examples, and a test set of 10,000 examples of handwritten digits. Each example is a 28 x 28 gray-level image. Some examples are depicted in Figure 3.4 in Chapter 3.

As before, I first divide the MNIST image database into training and test datasets. While the entire test set is retained for evaluation, nine data subsets are generated from the training dataset, varying from 1000 to 9000 training examples. There are two reasons for investigating the model behavior over these training sizes: (1) to examine consistency in GP-init recommendation, and (2) to conduct experiments within the limitations of available hardware.

Estimating hyperparameters from data

For each training dataset I estimate the design hyperparameter pair (σ_w^2, σ_b^2) by maximizing the ReLU Gaussian process log marginal likelihood. Since the Ground Truth hyperparameter pair associated with the MNIST data is unknown, I choose to explore the behavior of the models over the ranges $\sigma_w^2 \in [0.4, 1.2, 2.0, 2.8, 3.6]$ and $\sigma_b^2 \in [0, 1.0, 2.0]$. These ranges are selected based on the experimental values used in [20, 21]. As a result, fifteen models are constructed for every given test set, and with which the GP-init method produces consistent value of (3.6, 0.0) as the optimal hyperparameter pair for all MNIST training data subsets.

Prediction implementation details

As for the simulations, all neural network models for the prediction experiments are feedforward, fully-connected neural networks. The main difference among the models is in the number of hidden layers. Since the input dimension is 28 (pixels) x 28 (pixels) = 784, I choose 2000 nodes for each hidden-layer for training and testing. For model compilation, Adam optimizer is selected along with 'mse' loss function and 'accuracy' evaluation metric. Model training is then conducted with a batch size of 64 examples, and 200 passes of the complete training set. The other main difference is in model initialization.

Assessing GP-init on neural network prediction performance

For each training dataset generated with a specific training size, a neural network model is initialized with the estimated hyperparameter pair. A separate neural network model designed for evaluating He-init is initialized with (2.0, 0.0) prior to training.

The final step includes measuring and comparing prediction performance of GP-init against He-init benchmark. This is done by running the trained model on the test dataset corresponding to the depth of the model. For example, I evaluate a network model of depth $L = 3$ on the training and

test datasets generated using a recursive covariance function for depth $L = 3$.

Table 4.3: GP-init vs. He-init: comparing prediction accuracy on MNIST image dataset for 2- and 3-hidden layer network model.

Training Size	GP-init	He-Init	Best Case		Worst Case	
	Acc.	Acc.	Acc.	(σ_w^2, σ_b^2)	Acc.	(σ_w^2, σ_b^2)
L2: 1000	92.24	92.43	92.76	(0.4, 0)	92.07	(3.6, 0.1)
L2: 2000	93.84	93.53	93.85	(3.6, 0.1)	92.73	(0.4, 0.1)
L2: 3000	94.50	94.09	94.55	(3.6, 0.1)	93.53	(0.4, 0.1)
L2: 4000	94.81	94.31	94.91	(3.6, 0.1)	93.91	(0.4, 0.1)
L2: 5000	95.28	94.82	95.40	(3.6, 0.1)	94.45	(0.4, 0.1)
L2: 6000	95.86	95.54	95.94	(3.6, 0.1)	95.26	(0.4, 0.1)
L2: 7000	95.90	95.52	96.00	(3.6, 0.1)	95.23	(0.4, 0.1)
L2: 8000	95.96	95.61	96.05	(3.6, 0.1)	95.31	(0.4, 0.1)
L2: 9000	96.23	95.91	96.32	(3.6, 0.1)	95.65	(0.4, 0.1)
L3: 1000	92.24	92.36	92.38	(2, 0.01)	92.16	(3.6, 0.1)
L3: 2000	93.87	93.86	93.88	(1.2, 0.01)	93.79	(0.4, 0)
L3: 3000	94.53	94.71	94.78	(0.4, 0)	94.48	(3.6, 0.1)
L3: 4000	95.09	95.15	95.36	(0.4, 0)	95.06	(3.6, 0.1)
L3: 5000	95.46	95.53	95.70	(0.4, 0)	95.45	(2.8, 0)
L3: 6000	96.02	96.04	96.06	(0.4, 0)	95.99	(3.6, 0.01)
L3: 7000	96.21	96.24	96.33	(0.4, 0)	96.20	(3.6, 0.1)
L3: 8000	96.36	96.43	96.52	(0.4, 0.1)	96.35	(3.6, 0.1)
L3: 9000	96.46	96.52	96.71	(0.4, 0)	96.46	(3.6, 0)

Table 4.3 presents the prediction results, comparing GP-init to He-init, the best and the worst performer for various training sizes. The same results are also shown in Figure 3.5 for visual comparison, with the x-axis representing the training size and the y-axis the prediction accuracy.

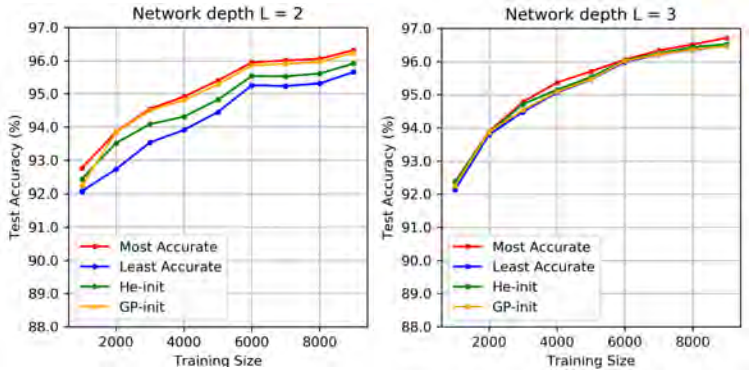

Figure 4.3: Comparing prediction accuracy of multi-hidden-layer network models on MNIST image dataset using GP-init vs. He-init methods.

4.4 Conclusion

The main objective of this chapter is to investigate if GP-initialization method based on Gaussian process optimization can improve model initialization in multi-hidden-layer neural networks and increase their prediction accuracy.

I present an alternative derivation of the recursive output covariance function of a multi-hidden-layer ReLU neural network. I apply the procedure of log marginal likelihood maximization in simulation experiments to study the effectiveness of the GP-init approach on neural network regression tasks, in comparison to the benchmark He-init method. The procedure is then used to assess GP-init on neural network image prediction with multilayer neural network models

The simulation experiments show that in general GP-init performs as well as if not better than He-init in regression prediction in either two- or three-hidden-layer models. The difference in performance between GP-init and He-init is more evident at $\sigma_b^2 = 1.0$. In addition, GP-init achieves about 50% of the He-init error rate. This suggests that both hyperparameters are important in the formulation of recursive covariance functions.

Furthermore, results from the MNIST prediction experiment indicate that GP-init improves prediction accuracy over He-init in a neural network with two hidden layers. However, the performance

difference between the two initialization methods becomes negligible in a three-hidden-layer network. On the other hand, consistency in GP-init recommendation (3.6, 0.0) for this experiment implies that smaller training sets could be used to provide reasonable recommendation for neural network initialization. This may help to reduce the computational time needed for inverting large covariance matrices resulting from training data.

In conclusion, the experiments conducted in this chapter demonstrate the efficacy and efficiency in applying GP-init in making recommendations for multi-hidden-layer neural network initialization. It might be interesting to find out if similar conclusions would result from other real-world datasets for different kinds of applications such as speech recognition.

Chapter 5

Neural Network Activation Function Selection

The significance of activation functions in neural networks and covariance functions in Gaussian processes is discussed in Chapter 2. My main goal in this chapter is to assess log marginal likelihood maximization technique in the selection of activation functions.

5.1 Monte Carlo Approximation

In Chapters 3 and 4, I demonstrate the techniques for obtaining covariance functions of Gaussian processes corresponding to single- and multi- hidden-layer neural networks with ReLU activation function. However, the list of standard activation functions is quite large [18], including ReLU, parametric ReLU, tanh, swish, and many more. Deriving closed-form covariance functions for some of these conventional activations might not be possible if they are analytically intractable. To address this issue, I turned to the Monte Carlo method.

Monte Carlo technique is a sampling method that can be used to approximate the empirical distribution, the median and the variance of certain parameters of a posterior distribution [46]. With a sufficiently large Monte Carlo sample, these quantities can be approximated to an arbitrary degree of precision. Generally, the method can be applied to estimate the expected value of a given random function [29].

For example, suppose the goal is to approximate the expectation of the function $\phi(W)$ where the random variable W has a density function $f_W(w)$:

$$E[\phi(W)] = \int_{-\infty}^{\infty} \phi(w) f_W(w)\, dw. \tag{5.1}$$

The Monte Carlo estimation method starts with drawing a set of samples $w^{(m)}, m \in \{1, 2, \cdots, M\}$, independently from the distribution $f_W(w)$. Then, the expected value in Equation (5.1) can be

approximated by the sum

$$\bar{\phi} = \frac{1}{M} \sum_{m=1}^{M} \phi(w^{(m)}). \tag{5.2}$$

As an illustration, consider a single-hidden-layer neural network with activation function $\phi(x)$. As shown in Chapter 3, the output covariance function of the network is given by

$$\mathcal{C}(x^0, y^0) \coloneqq E[(b_i^1)^2] + \sum_{j=1}^{N_1} E[(W_{ij}^1)^2] E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)]. \tag{5.3}$$

Now I wish to apply the Monte Carlo method to approximate the quantity

$$\begin{aligned} & E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)] \\ = & \int \cdots \int_{-\infty}^{\infty} \phi(b_j^0 + w_j^0 \cdot x^0)\phi(b_j^0 + w_j^0 \cdot y^0) f_{B_j^0, W_j^0}(b_j^0, w_j^0)\, dw_j^0\, db_j^0. \end{aligned} \tag{5.4}$$

Conventional approach

First, values of $w^{(m)}$ and $b^{(m)}$ are sampled independently from the distributions $f_W(w)$ and $f_B(b)$. This allows the approximate value of Equation (5.4) to be computed as

$$\bar{\phi}(x^0, y^0) = \frac{1}{M} \sum_{m=1}^{M} \phi(b^{(m)} + w^{(m)} \cdot x^0)\phi(b^{(m)} + w^{(m)} \cdot y^0). \tag{5.5}$$

In the limit of $M \to \infty, \bar{\phi}(x^0, y^0) \to E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)]$. In other words, the Monte Carlo approximation $\bar{\phi}(x^0, y^0)$ approaches the kernel $E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)]$ as the number of samples increases. A close estimate of the output covariance function, $\mathcal{C}(x^0, y^0)$ in Equation 5.3, is obtained when the sample size is large.

This is a straightforward approach for obtaining Monte Carlo approximation to the covariance function. However, as the dimension of input data increases, the number of multiplications within the activation functions ϕ increases. For large input dimensions, an alternative approach using the U-V transformation introduced in Appendix A.1 is proposed to reduce computational cost.

A modified approach

In general, the arguments of the activation functions in Equation (5.4) are pre-activations that can be written as

$$U = b_j^0 + W_j^0 \cdot x^0 \sim \mathcal{N}(0, \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2), V = b_j^0 + W_j^0 \cdot y^0 \sim \mathcal{N}(0, \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2).$$

This gives the joint Gaussian distribution of the random variables U, V:

$$\begin{pmatrix} U \\ V \end{pmatrix} \sim \mathcal{N}(0, \Sigma), \text{ where } \Sigma = \begin{pmatrix} \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2 & \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y) \\ \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y) & \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2 \end{pmatrix}.$$

Equation (5.4) can then be rewritten as

$$\begin{aligned} &\iint_{-\infty}^{\infty} \phi(u)\phi(v) \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} \exp\big(-\frac{1}{2}(u,v)\Sigma^{-1}(u,v)^T\big)\, du\, dv \\ &= \iint_{-\infty}^{\infty} \phi(u)\phi(v) f_{U,V}(u,v)\, du\, dv \end{aligned} \tag{5.6}$$

Now, values of $u^{(m)}$ and $v^{(m)}$ are sampled independently from their joint distributions $f_{U,V}(u,v)$. The approximate value of Equation (5.6) to be computed as

$$\bar{\phi}_{UV}(x^0, y^0) = \frac{1}{M}\sum_{m=1}^{M} \phi(u)^m \phi(v)^m. \tag{5.7}$$

In the limit of $M \to \infty, \bar{\phi}_{UV}(x^0, y^0) \to E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)]$. A close estimate of the output covariance function can then be obtained.

The hypothesis is that this modified approach may improve efficiency in computing Monte Carlo approximation of a desired covariance function. Future experiments will be needed to verify the assertion.

5.1.1 Empirical validation: A Gaussian process regression example

I now conduct a simulation to empirically validate the Monte Carlo method in approximating the exact output covariance function corresponding to a ReLU activation.

The simulation experiment consists of three steps. First, simulated data are generated by using a Gaussian process model with ReLU covariance function and hyperparameter pair $(\sigma_w^2, \sigma_b^2) = (2.8, 0.001)$. 10 sample paths are produced from which one is randomly chosen to provide 70 training and 30 test points, as depicted in Figure 5.1.

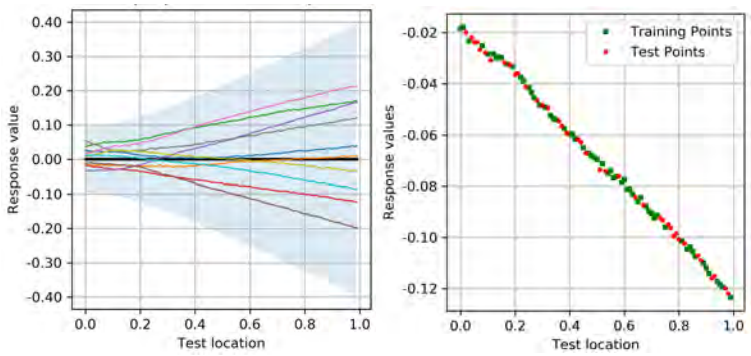

Figure 5.1: Sample paths and test points for empirical validation of Monte Carlo method for approximating ReLU neural network output covariance function.

Next, a Gaussian process regression model is built with the exact ReLU covariance function to make predictions on the previously unseen test points. A second Gaussian process regression model is concurrently built with the Monte Carlo approximation to the ReLU covariance function to make predictions on the same test set. The models are trained with the training dataset and then evaluated on the test set.

The comparison between predictions resulting from the exact form and its Monte Carlo approximation, respectively, is presented in Figure 5.2. The pairwise response comparison on the left panel and the line of equality comparison on the right panel support the notion that utilizing Monte Carlo approximation in place of the analytic form of covariance function is reasonable.

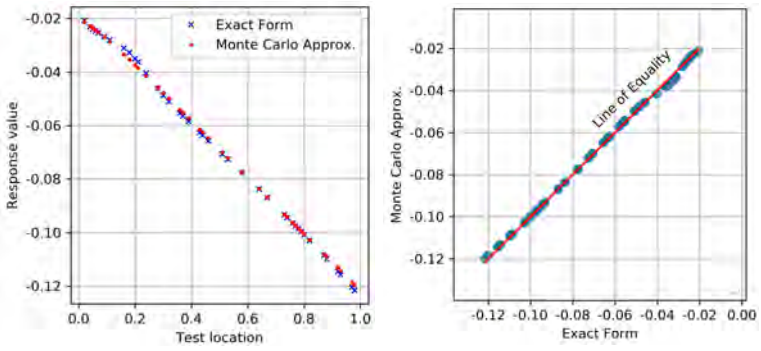

Figure 5.2: Empirical validation of Monte Carlo method for approximating ReLU neural network output covariance function.

5.2 Activation Function Selection

5.2.1 Simulation objective

In a typical setting, ReLU activation function is chosen as the nonlinearity for neural network models. It is also a common practice to initialize models with He-initialization which specifies the hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{He} = \left(\frac{2}{n_l}, 0\right)$ where n_l is the hidden layer width.

The simulation conducted in this section aims to demonstrate that the log marginal likelihood maximization technique is able to select a more appropriate activation function and hyperparameter pairs to improve neural network prediction accuracy. Accordingly, a dataset generated with logistic rather than ReLU covariance function is chosen for the experiment. The logistic function is defined as

$$\phi(x) \coloneqq \frac{1}{1+e^{-x}}$$

whose Monte Carlo approximation can be obtained by following the steps outlined in Section 5.1. The exact expression for ReLU covariance function is provided in Chapter 3.

There are two main parts in the simulation experiment. First, hyperparameter pairs are estimated for the Gaussian process model with logistic covariance function, $(\sigma_w^2, \sigma_b^2)_{logistic}$, and for the Gaussian process model with ReLU covariance function, $(\sigma_w^2, \sigma_b^2)_{ReLU}$. Their corresponding marginal

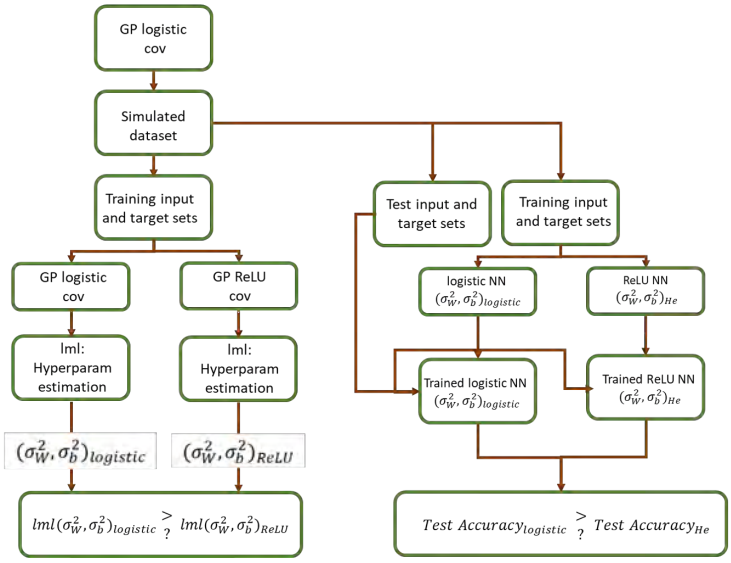

Figure 5.3: Activation function selection based on log marginal likelihood maximization.

likelihoods are computed for comparison. The second step is evaluating the prediction accuracy of the neural network model configured with logistic activation function, initialized using the estimated hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{logistic}$, and a ReLU neural network initialized using He-initialization method. The flowchart of the procedure is shown in Figure 5.3.

5.2.2 Data generation

I first generate simulation data using a Gaussian process model with logistic covariance function and design hypermarameter pair $(\sigma_w^2, \sigma_b^2)_{design} = (0.5, 1.0)$, as suggested in Chapter 3. The dataset is randomly split into training and test sets. The training set is then applied to train the logistic Gaussian process model to learn from data the hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{logistic}$.

5.2.3 Selecting activation functions with log marginal likelihood maximization

The hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{logistic} = (0.008, 0.0)$ is learned via log marginal likelihood maximization and identified over the search grids: $\sigma_w^2 \in [0.008, 0.01, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 1.0, 2.0]$; $\sigma_b^2 \in [0.0, 0.05, 0.1, 0.5, 1.0, 2.0]$.

The first part of the experiment shows that the log marginal likelihood obtained with the logistic Gaussian process model is larger than that with the ReLU model, as presented in Table 5.1. This result picks logistic activation function to be used for neural network prediction in part 2..

Table 5.1: Logistic vs. ReLU: model log marginal likelihood on Logistic dataset.

Metric	Logistic GP-init Model		ReLU He-init Model	
	lml	$(\sigma_w^2, \sigma_b^2)_{logistic}$	lml	$(\sigma_w^2, \sigma_b^2)_{ReLU}$
Log marginal likelihood	-120.70	(0.008 0.00)	-1226.49	(2.0, 0.00)

5.2.4 Validating the likelihood maximization approach

The second part of the simulation trains and evaluates ReLU and logistic neural networks. Two single-hidden-layer neural networks each with 100 hidden nodes were built. The neural network model with logistic activation function is initialized with the learned hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{logistic} = (0.008, 0.0)$. A second model is configured according to conventional practice: with ReLU activation function and initialized with the hyperparameter pair $(\sigma_w^2, \sigma_b^2)_{He} = (2.0, 0.0)$. Each of the models is run 50 times to collect prediction mean-square-errors.

5.2.5 Neural network prediction

The simulation results are shown in Table 5.2. They indicate that the GP-init method not only learns a better hyperparameter pair than the fixed $(2.0, 0.0)$ suggested by He-init, but also selects a more appropriate activation function than the standard ReLU for the application and dataset. The method estimates the hyperparameter pair from data through log marginal likelihood maximization and chooses logistic activation function that enables the neural network to achieve higher prediction accuracy and lower variance.

Furthermore, the boxplot in Figure 5.4 displays a much narrower interquartile range in the outcome

using GP-init scheme, indicating that the proposed GP-init method is more consistent than the benchmark He-initialization method.

Table 5.2: GP-init vs. He-init: Neural network regression error (MSE) on Logistic dataset.

	Proposed Method	Standard Method	
	Logistic GP-init Model	ReLU He-init Model	Selection
Mean	1.47	2.29	GP-init
Median	1.47	1.68	GP-init

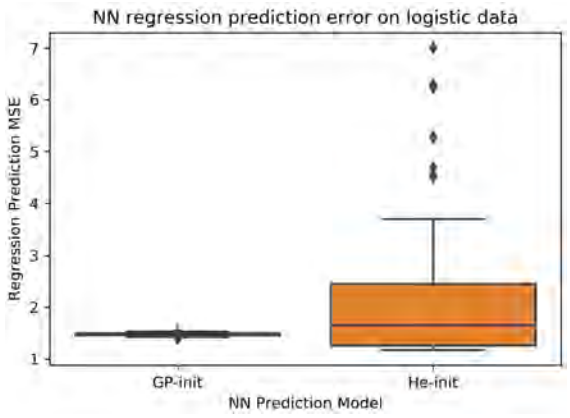

Figure 5.4: NN regression prediction error on logistic dataset.

5.3 Analysis and Conclusion

I describe the procedure for applying Monte Carlo method to approximate covariance functions that may be too complex to derive. To examine the preciseness of the method, I conduct a simulation where the Monte Carlo approximation of ReLU covariance function is compared to the exact expression in a regression task. The experiment shows that prediction results obtained from the Monte Carlo method closely approximate those from the exact expression, thus validating the approximation method.

The main objective of this section is to determine how marginal likelihood maximization can be utilized to select activation function for neural network models. In practice a neural network model

applies ReLU activation function and is initialized with He-initialization method prior to training. I design and conduct a simulation experiment to test GP-init against He-init method in choosing an appropriate activation function given the training and test data.

Empirical results show that the GP-init method learns a hyperparameter pair and selects an activation function for the neural network model that achieves higher prediction accuracy than the neural network with the same structural design but with ReLU activation function, and initialized using He-init method. In addition, the accuracy of the neural network model trained based on GP-init is more consistent than the model trained with the standard ReLU activation function and He-initialization.

Chapter 6

Summary and Future Directions

6.1 Conclusion

This dissertation addresses the research question: Is there a statistical approach that can learn from data to improve neural network model initialization and activation function selection?

Proper model initialization and appropriate activation functions are critical to neural network training. However, existing initialization schemes do not consider extracting relevant information from data. In addition, it is common to use ReLU as the standard activation function for prediction tasks. In this work, I show that Gaussian process optimization offers a tool, marginal likelihood maximization, that learns model hyperparameters from data and improves neural network initialization. The method of using marginal likelihood maximization, known as GP-init in this dissertation, also guides the selection of proper activation function in neural networks.

To evaluate the efficacy of GP-init, I design and conduct experiments on simulated data and MNIST hand-written digit dataset. Hyperparameters are first learned via computing marginal likelihood. Single-hidden-layer neural network models are then initialized with the learned hyperparameters prior to model training. After training, the networks are evaluated over test datasets and compared to the benchmark He-initialization method. Empirical results suggest that GP-init achieved better prediction accuracy than He-init for input data characterized by small weight variance and large bias variance. Furthermore, both weight and bias parameters in a neural network architecture are essential for obtaining the covariance function of its corresponding Gaussian process.

Similar experiments are performed on multilayer neural network models. Results from simulation show that GP-init generally performs better than He-init in prediction accuracy. This was particularly evident for input data characterized by small weight variance and large bias variance, as

for the single-hidden-layer model. Results from MNIST prediction experiment also indicate that GP-init improves prediction accuracy over He-init in a neural network with two hidden layers. However, their performance difference became very small in a three-hidden-layer network.

As mentioned in the beginning of this chapter, in practice a neural network model usually applies the de facto standard ReLU activation function. I design and conduct a simulation on a single-hidden-layer neural network to test GP-init in choosing appropriate activation functions. Empirical results show that GP-init is able to choose activation function in accordance with data. Using the selected activation function and the learned hyperparameters for initialization, the neural network model is shown to achieve higher prediction accuracy than the model using ReLU activation function and the He-initialization method.

All in all, GP-init seems to be a promising technique for improving neural network model initialization and activation function selection.

6.2 Future Directions

Future research efforts are needed to further measure the validity and reliability of the GP-init method. They include performing additional experiments with activation functions other than ReLU and logistic activations. A notable candidate is the parametric ReLU (PReLU). It is a learned parametric activation that generalizes the standard rectified linear unit [4]. Another candidate is the Swish activation function [8, 9].

In this dissertation, only fully-connected, feedforward neural networks are considered. Future work will examine architectures such as convolutional neural networks [47], recurrent neural networks [48], and residual networks [49].

Another area for future research is derivation of closed-form covariance functions corresponding to other standard activation functions. Even though Monte Carlo Approximation may be useful for estimating covariance functions that may not be analytically tractable, it is not ideal. Finding the

closed-form expression should still be considered whenever it may be possible. Parametric ReLU (PReLU) is a desirable candidate.

Finally, future research also includes incorporating GP-init into investigating the maximum trainable depth of a neural network, in comparison to the notion of edge of chaos [20] where the depth of a trainable deep network is dependent on the choice of model hyperparameters.

6.3 Contributions

Recent research efforts in machine learning expand the equivalence of single-hidden-layer neural networks and Gaussian process models [10] to fully-connected deep networks [23] and deep convolutional neural networks [38]. In conjunction with the work in developing kernel methods for deep learning [11], they provided inspiration and motivation for this dissertation.

There are three contributions in this dissertation. First, I investigated the link between neural networks and Gaussian processes to explore statistical methods for initializing neural networks and choosing activation functions. I discovered that log marginal likelihood maximization could indeed be helpful for improving neural network initialization. Second, I designed and implemented Gaussian process and neural network models to test and evaluate the GP-initialization method. Experiments on simulated data and real-world image dataset indicated that log marginal likelihood maximization is a promising approach for network initialization. Lastly, I demonstrated that GP-initialization was also effective in selecting activation functions in neural network models. This is important as properly chosen activation functions can enhance neural network training and help improve model predictive accuracy.

Bibliography

[1] Santhosh K. Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy Anticipation for Efficient Exploration and Navigation. *arXiv:2008.09285 [cs]*, August 2020. arXiv: 2008.09285.

[2] Geeticka Chauhan, Ruizhi Liao, William Wells, Jacob Andreas, Xin Wang, Seth Berkowitz, Steven Horng, Peter Szolovits, and Polina Golland. Joint Modeling of Chest Radiographs and Radiology Reports for Pulmonary Edema Assessment. *arXiv:2008.09884 [cs]*, August 2020. arXiv: 2008.09884.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, page 8, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, February 2015. arXiv: 1502.01852.

[5] Dmytro Mishkin and Jiri Matas. All you need is a good init. *2016 ICLR*, February 2016. arXiv: 1511.06422.

[6] Arnu Pretorius, Elan Van Biljon, S. Kroon, and H. Kamper. Critical initialisation for deep signal propagation in noisy rectifier neural networks. In *NeurIPS*, 2018.

[7] Rebekka Burkholz and Alina Dubatovka. Initialization of relus for dynamical isometry. *Advances in Neural Information Processing Systems*, 32:2385–2395, 2019.

[8] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. *arXiv:1710.05941 [cs]*, October 2017. arXiv: 1710.05941 version: 1.

[9] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the Selection of Initialization and Activation Function for Deep Neural Networks. *arXiv:1805.08266 [cs, stat]*, October 2018. arXiv: 1805.08266.

[10] Radford M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Springer New York, New York, NY, 1996.

[11] Youngmin Cho and Lawrence K. Saul. Kernel Methods for Deep Learning. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 342–350. Curran Associates, Inc., 2009.

[12] Christopher K. I. Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT Press, 2006.

[13] G. Cybenko. Approximation by superpositions of a sigmoidal function. *1989 Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

[14] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, January 1989.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[17] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[18] Activations TensorFlow. Module: tf.keras.activations | TensorFlow Core v2.4.1.

[19] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[20] Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep Information Propagation. *ICLR 2017*, April 2017. arXiv: 1611.01232.

[21] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the Impact of the Activation Function on Deep Neural Networks Training. *ICML*, May 2019. arXiv: 1902.06853.

[22] Yann LeCun. THE MNIST DATABASE of handwritten digits, 1998.

[23] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. *2018 ICLR*, March 2018. arXiv: 1711.00165.

[24] Douglas C. Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[25] John W. Tukey. *Exploratory data analysis*, volume 2. Reading, Mass., 1977.

[26] David C. Hoaglin, Frederick Mosteller, and John W. Tukey. *Understanding robust and exploratory data analysis*. Number Sirsi) i9780471384915. 2000.

[27] Jacquelyn Bulao. How Much Data Is Created Every Day in 2020?, January 2021.

[28] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[29] Christopher M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[30] David MacKay. Introduction to Gaussian processes. *Citeseer*, 1998.

[31] Radford M. Neal. Regression and classification using Gaussian process priors. *1998 Bayesian statistics*, 6:475, 1998.

[32] Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for machine learning, 2006.

[33] Raquel Urtasun and Trevor Darrell. Sparse probabilistic regression for activity-independent human pose inference. *CVPR*, 2008.

[34] Ashish Kapoor, Kristen Grauman, Raquel Urtasun, and Trevor Darrell. Gaussian Processes for Object Categorization. *International Journal of Computer Vision*, 88(2):169–188, June 2010.

[35] Mark van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian Processes. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2849–2858. Curran Associates, Inc., 2017.

[36] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[37] David Duvenaud, Oren Rippel, Ryan P Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. *AISTATS*, page 9, 2014.

[38] Adrià Garriga-Alonso, Carl Edward Rasmussen, and Laurence Aitchison. Deep Convolutional Networks as shallow Gaussian Processes. *ICLR*, May 2019. arXiv: 1808.05587.

[39] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, September 2009. ISSN: 2380-7504.

[40] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *2010 ICML*, page 8, 2010.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[42] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013. Issue: 1.

[43] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton. On rectified linear units for speech processing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521, May 2013. ISSN: 2379-190X.

[44] Youngmin Cho. *Kernel Methods for Deep Learning*. PhD thesis, UC - SAN DIEGO, 2012.

[45] Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215. JMLR, 2013.

[46] Peter D. Hoff. *A first course in Bayesian statistical methods*, volume 580. Springer, 2009.

[47] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. Publisher: Ieee.

[48] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv:1504.00941 [cs]*, April 2015. arXiv: 1504.00941.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016. IEEE.

[50] Izrael S. Gradshteyn. *Table of integrals*. Elsevier Science, 2014.

Appendix A

Appendix

A.1 Covariance Function for Single-hidden-layer ReLU Neural Networks

Denote the input layer (layer 0) weight and bias parameters as $b_j^0 \sim \mathcal{N}(0, \sigma_b^2)$ and $W_{jk}^0 \overset{\text{iid}}{\sim} \mathcal{N}(0, \frac{\sigma_w^2}{d_{in}})$, where $b_j^0 \perp\!\!\!\perp W_{jk}^0$ for all $k \in \{1, \cdots, d_{in}\}, j \in \{1, \cdots, N_1\}$.

The expected value of the product of post-activations at the output of the j^{th} hidden node is computed as

$$\begin{aligned}
&E[\mathbf{X}_j(x^0)\mathbf{X}_j(y^0)] \\
&= \int \cdots \int_{-\infty}^{\infty} \max(b_j^0 + w_j^0 \cdot x^0)\, \max(b_j^0 + w_j^0 \cdot y^0) f_{b_j^0, W_j^0}(b, w)\, dw_j^0\, db_j^0 \\
&= \int \cdots \int_{-\infty}^{\infty} (b_j^0 + w_j^0 \cdot x^0)_+ (b_j^0 + w_j^0 \cdot y^0)_+ f_{b_j^0, W_j^0}(b, w)\, dw_j^0\, db \qquad \text{(A.1)}
\end{aligned}$$

Each pre-activation can be written in terms of a random variable:

$$\begin{aligned}
U = b_j^0 + W_j^0 \cdot x^0 = b_j^0 + \sum_{k=1}^{d_{in}} W_{jk}^0 x_k^0 \sim \mathcal{N}(0, \sigma_b^2 + \frac{\sigma_w^2}{d_{in}} \|x\|^2), \\
V = b_j^0 + W_j^0 \cdot y^0 = b_j^0 + \sum_{k'=1}^{d_{in}} W_{jk'}^0 y_{k'}^0 \sim \mathcal{N}(0, \sigma_b^2 + \frac{\sigma_w^2}{d_{in}} \|y\|^2).
\end{aligned}$$

Since $E[U] = E[V] = 0$, their covariance can be expressed as

$$\begin{aligned}
&\text{cov}(U, V) = E[(b_j^0 + W_j^0 \cdot x^0)(b_j^0 + W_j^0 \cdot y^0)] \\
&= E[(b_j^0)^2] + E\Big[\sum_{k=1}^{d_{in}} \sum_{k'=1}^{d_{in}} W_{jk}^0 W_{jk'}^0 x_k^0 y_{k'}^0\Big] \\
&= \sigma_b^2 + \sum_{k=1}^{d_{in}} \sum_{k'=1}^{d_{in}} E\Big[W_{jk}^0 W_{jk'}^0\Big] x_k^0 y_{k'}^0 \\
&= \sigma_b^2 + \frac{\sigma_w^2}{d_{in}} \sum_{k=1}^{d_{in}} x_k^0 y_k^0 \\
&= \sigma_b^2 + \frac{\sigma_w^2}{d_{in}} (x \cdot y) \quad \text{(For simplicity } x = x^0, y = y^0)
\end{aligned}$$

This implies that the random variables U, V have a joint Gaussian distribution:

$$\begin{pmatrix} U \\ V \end{pmatrix} \sim \mathcal{N}(0, \Sigma), \text{ where } \Sigma = \begin{pmatrix} \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2 & \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y) \\ \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y) & \sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2. \end{pmatrix}$$

Equation (A.1) can therefore be rewritten as

$$\iint_0^\infty uv \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} \exp\big(-\frac{1}{2}(u,v)\Sigma^{-1}(u,v)^T\big)\, du\, dv \tag{A.2}$$

Denote $D \coloneqq |\Sigma| = \big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2\big)\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2\big) - \big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)^2$, and

$$\Sigma^{-1} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

with $a_{11} = \frac{1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2)$, $a_{22} = \frac{1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2)$, and
$a_{12} = a_{21} = \frac{-1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y))$.

This leads to:

$$\begin{aligned} & D(a_{11}a_{22} - a_{12}^2) \\ =& D\Big(\frac{1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2)\frac{1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2) - \big(\frac{-1}{D}(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)^2\Big) \\ =& \frac{1}{D}\Big(\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2\big)\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2\big) - \big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)^2\Big) \\ =& 1. \quad \text{(by definition)} \end{aligned} \tag{A.3}$$

The exponential term in equation (A.2) then becomes:

$$-\frac{1}{2}(u,v)\Sigma^{-1}(u,v)^T = -\frac{1}{2}\big(a_{11}u^2 + 2a_{12}uv + a_{22}v^2\big).$$

Transformation from Cartesian to polar coordinates is performed by setting

$$u = \frac{r}{\sqrt{a_{11}}} \cos\alpha,\ v = \frac{r}{\sqrt{a_{22}}} \sin\alpha$$

$$\implies a_{11}u^2 = r^2\cos^2\alpha,\ a_{22}v^2 = r^2\sin^2\alpha.$$

The Jacobian $\mathcal{J}$ is calculated as

$$\Big|\frac{\partial(u,v)}{\partial(r,\alpha)}\Big| = \frac{r}{\sqrt{a_{11}a_{22}}}.$$

Equation (A.2) can in turn be expressed as

$$\frac{1}{2\pi\mathcal{D}^{1/2}}\int_{\alpha=0}^{\frac{\pi}{2}}\int_{r=0}^{\infty}\frac{r^2\,\sin\,2\alpha}{2\sqrt{a_{11}a_{22}}}\exp\Big(\frac{-1}{2}[r^2\cos^2\alpha+\frac{2a_{12}r^2\,\sin\,\alpha\,\cos\,\alpha}{\sqrt{a_{11}a_{22}}}+r^2\sin^2\alpha]\Big)\frac{r\,dr\,d\alpha}{\sqrt{a_{11}a_{22}}}$$
$$=\frac{1}{4\pi\mathcal{D}^{1/2}a_{11}a_{22}}\int_{\alpha=0}^{\frac{\pi}{2}}\,\sin\,2\alpha\,d\alpha\int_{r=0}^{\infty}r^3\exp\Big(\frac{-r^2}{2}[1+\frac{a_{12}\,\sin\,2\alpha}{\sqrt{a_{11}a_{22}}}]\Big)\,dr \tag{A.4}$$

Next, it is necessary to show that $\mathcal{H}\coloneqq 1+\frac{a_{12}\,\sin\,2\alpha}{\sqrt{a_{11}a_{22}}}\geq 0$ to ensure the expression in (A.4) is bounded.

First, since $\|x-y\|^2=\|x\|^2+\|y\|^2-2(x\cdot y)\geq 0\implies\|x\|^2+\|y\|^2\geq 2(x\cdot y)$, and let the angle between the vectors x,y be $\theta=\cos^{-1}\Big(\frac{x\cdot y}{\|x\|\|y\|}\Big)$,

$$\begin{aligned}&\frac{\Big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}(x\cdot y)\Big)^2}{\Big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|x\|^2\Big)\Big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|y\|^2\Big)}\\&=\frac{\sigma_b^4+(\frac{\sigma_w^2}{d_{in}})^2(x\cdot y)^2+2\sigma_b^2(\frac{\sigma_w^2}{d_{in}})(x\cdot y)}{\sigma_b^4+(\frac{\sigma_w^2}{d_{in}})^2\big(\|x\|^2\|y\|^2\big)+\sigma_b^2(\frac{\sigma_w^2}{d_{in}})\big(\|x\|^2+\|y\|^2\big)}\\&=\frac{\sigma_b^4+(\frac{\sigma_w^2}{d_{in}})^2(\|x\|\|y\|\,\cos\,\theta)^2+\sigma_b^2(\frac{\sigma_w^2}{d_{in}})2(x\cdot y)}{\sigma_b^4+(\frac{\sigma_w^2}{d_{in}})^2\big(\|x\|^2\|y\|^2\big)+\sigma_b^2(\frac{\sigma_w^2}{d_{in}})\big(\|x\|^2+\|y\|^2\big)}\qquad\leq 1.\end{aligned}$$

This means that one can define a quantity ϕ as

$$\begin{aligned}\phi&=\cos^{-1}\frac{\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}(x\cdot y)\big)}{\Big((\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)^{1/2}}\\&=\cos^{-1}\frac{\frac{1}{D}\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}(x\cdot y)\big)}{\frac{1}{D}\Big((\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)}\\&=\cos^{-1}\Big(\frac{-a_{12}}{\sqrt{a_{11}a_{22}}}\Big)\\&\implies\cos\,\phi=\Big(\frac{-a_{12}}{\sqrt{a_{11}a_{22}}}\Big)\end{aligned}\tag{A.5}$$

This also leads to

$$\mathcal{H}\coloneqq 1+\frac{a_{12}\,\sin\,2\alpha}{\sqrt{a_{11}a_{22}}}$$

$$
\begin{aligned}
&= 1 + \frac{\frac{-1}{D}\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)\ \sin 2\alpha}{\frac{1}{D}\Big((\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)} \\
&= 1 - \frac{\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)\ \sin 2\alpha}{\Big((\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)^{1/2}} \\
&\geq 1 - \frac{\big(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}(x \cdot y)\big)}{\Big((\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2 + \frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)^{1/2}} \\
&\geq 0 \qquad \blacksquare
\end{aligned}
$$

With a change of variables, one can now evaluate the integral involving the parameter r in expression (A.4) as follows.

Let $\eta = \frac{r^2}{2}\mathcal{H}$. Then $r = \sqrt{\frac{2\eta}{\mathcal{H}}} \implies d\,r = \frac{1}{2}\sqrt{\frac{2}{\mathcal{H}}}\eta^{-1/2}\,d\,\eta$.

$$
\begin{aligned}
&\int_{\eta=0}^{\infty} \Big(\frac{2}{\mathcal{H}}\Big)^{\frac{3}{2}} \eta^{\frac{3}{2}} \mathrm{e}^{-\eta} \frac{1}{2}\sqrt{\frac{2}{\mathcal{H}}}\eta^{\frac{-1}{2}}\,d\eta \\
&= \int_{\eta=0}^{\infty} \frac{2^2}{\mathcal{H}^2}\frac{1}{2}\eta \mathrm{e}^{-\eta}\,d\eta \\
&= \frac{2}{\mathcal{H}^2}\int_{\eta=0}^{\infty} \eta\,\mathrm{e}^{-\eta}\,d\eta \\
&= \frac{2}{\mathcal{H}^2}\Gamma(2) = \frac{2}{\mathcal{H}^2} \\
&= \frac{2}{\Big(1 + \frac{a_{12}\ \sin 2\alpha}{\sqrt{a_{11}a_{22}}}\Big)^2} \\
&= \frac{2}{\Big(1 - \cos\,\phi\ \sin 2\alpha\Big)^2}. \qquad \big(\text{from Equation A.5}\big)
\end{aligned}
$$

The complete expression (A.4) becomes

$$
\frac{1}{4\pi\mathcal{D}^{1/2}a_{11}a_{22}}\int_{\alpha=0}^{\frac{\pi}{2}} \frac{2\ \sin 2\alpha}{\Big(1 - \cos\,\phi\ \sin 2\alpha\Big)^2}\,d\alpha \tag{A.6}
$$

$$
= \frac{1}{2\pi\mathcal{D}^{1/2}a_{11}a_{22}\sin^3\phi}\Big(\ \sin(\phi) + (\pi - \phi)\ \cos(\phi)\Big). \qquad \big(\text{from Equation A.9}\big)
$$

where $\phi = \cos^{-1}\Big(\frac{-a_{12}}{\sqrt{a_{11}a_{22}}}\Big)$.

Finally,

$$
2\pi\mathcal{D}^{1/2}a_{11}a_{22}\sin^3\phi
$$

$$
\begin{aligned}
&= 2\pi\mathcal{D}^{1/2}a_{11}a_{22}\,(1-\cos^2\phi)^{3/2} \\
&= 2\pi\mathcal{D}^{1/2}a_{11}a_{22}\,\Big(1-\frac{a_{12}^2}{a_{11}a_{22}}\Big)^{3/2} \\
&= 2\pi\mathcal{D}^{1/2}\Big(a_{11}a_{22}\Big)^{-1/2}\,\Big(a_{11}a_{22}-a_{12}^2\Big)^{3/2} \\
&= 2\pi\Big(\mathcal{D}^2a_{11}a_{22}\Big)^{-1/2}\,\Big(\mathcal{D}\big(a_{11}a_{22}-a_{12}^2\big)\Big)^{3/2} \\
&= 2\pi\Big(\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|x\|^2\big)\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|y\|^2\big)\Big)^{-1/2}\Big(1\Big)^{3/2}) \qquad \text{(A.7)}
\end{aligned}
$$

The expected value of the product of post-activations at the output of the j^{th} hidden node in the first hidden layer is therefore determined to be

$$
\begin{aligned}
&E[\mathbf{X}_j(x)\mathbf{X}_j(y)] \\
&= \frac{1}{2\pi}\Big(\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|x\|^2\big)\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|y\|^2\big)\Big)^{1/2}\Big(\ \sin(\phi)+(\pi-\phi)\ \cos(\phi)\Big), \\
&\text{where } \phi=\cos^{-1}\left\{\frac{\big(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}(x\cdot y)\big)}{\Big((\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|x\|^2)(\sigma_b^2+\frac{\sigma_w^2}{d_{in}}\|y\|^2)\Big)^{1/2}}\right\}.
\end{aligned}
$$

The covariance function at the network output is therefore determined to be

$$
\begin{aligned}
&E\Big[\big(b_i^1+\sum_{j=1}^{N_1}W_{ij}^1\mathbf{X}_j(x)\big)\big(b_i^1+\sum_{k=1}^{N_1}W_{ik}^1\mathbf{X}_k(y)\big)\Big]-E\Big[b_i^1+\sum_{j=1}^{N_1}W_{ij}^1\mathbf{X}_j(x)\Big]\Big[b_i^1+\sum_{k=1}^{N_1}W_{ik}^1\mathbf{X}_k(y)\Big] \\
&= E[(b_i^1)^2]+\sum_{j=1}^{N_1}E[(W_{ij}^1)^2]E[\mathbf{X}_j(x)\mathbf{X}_j(y)] \\
&= \sigma_b^2+\frac{\sigma_w^2}{N_1}N_1E[\mathbf{X}_j(x)\mathbf{X}_j(y)] \\
&= \sigma_b^2+\frac{\sigma_w^2}{2\pi}\Big(\sigma_b^2+\|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2+\|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sin\,\phi+(\pi-\phi)\,\cos\,\phi\Big). \qquad \blacksquare
\end{aligned}
$$

Remarks:

- The derivation follows the work on arc-cosine family of kernels developed in [11]. However, instead of applying coplanar vector rotation in calculating the kernel integral, it is recognized that the integrand can be written in terms of two jointly normal random variables. This helps to facilitate the computation which becomes more involved when both the weight and bias parameters are included.

- The derivation is also made to conform to the arc-cosine kernel by utilizing the identities [11, equation (17), (18)] to give us

$$\int_{\eta=0}^{\frac{\pi}{2}} \frac{1}{1 - \cos\,\phi\,\cos\,\eta}\, d\eta = \frac{\pi - \phi}{\sin\,\phi}, \tag{A.8}$$

$$\int_{\theta=0}^{\frac{\pi}{2}} \frac{\sin\,2\theta}{\big(1 - \cos\,\phi\,\sin\,2\theta\big)^2}\, d\theta$$

$$= \frac{1}{\sin^3\phi}\Big(\sin(\phi) + (\pi - \phi)\,\cos(\phi)\Big). \tag{A.9}$$

- Equation (A.9) is derived with the substitution $\eta = 2(\theta - \frac{\pi}{4})$ as follows:

$$\int_{\theta=0}^{\frac{\pi}{2}} \frac{\sin\,2\theta}{\big(1 - \cos\,\phi\,\sin\,2\theta\big)^2}\, d\theta$$

$$= \int_{\eta=0}^{\frac{\pi}{2}} \frac{\cos\,\eta}{\big(1 - \cos\,\phi\,\cos\,\eta\big)^2}\, d\eta$$

$$= \frac{\partial}{\partial \cos\,\phi} \int_{\eta=0}^{\frac{\pi}{2}} \frac{1}{1 - \cos\,\phi\,\cos\,\eta}\, d\eta$$

$$= \frac{\partial}{\partial \cos\,\phi}\Big(\frac{\pi - \phi}{\sin\,\phi}\Big) = \frac{-1}{\sin(\phi)} \frac{\partial}{\partial \phi}\Big(\frac{\pi - \phi}{\sin\,\phi}\Big)$$

$$= \frac{1}{\sin^3\phi}\Big(\sin(\phi) + (\pi - \phi)\,\cos(\phi)\Big). \qquad \blacksquare$$

- Using Chain rule, I obtain

$$\frac{\partial}{\partial \cos\phi} = \frac{\partial}{\partial \phi} \cdot \frac{\partial \phi}{\partial \cos\phi} = \frac{\partial}{\partial \phi} \Big/ \frac{\partial \cos\phi}{\partial \phi} = \frac{1}{-\sin\phi} \cdot \frac{\partial}{\partial \phi}.$$

- An alternative way for proving Equation A.8 is by applying formula **2.553** 3^* in [50]:

$$\int \frac{dx}{a + b\cos x} = \frac{2}{\sqrt{a^2 - b^2}} \arctan\Big(\frac{(a-b)\tan(\frac{x}{2})}{\sqrt{a^2 - b^2}}\Big) \tag{A.10}$$

$$\text{for } \Big[a^2 > b^2\Big].$$

Setting $a = 1, b = -\cos\phi$, I have

$$\int_{\eta=0}^{\frac{\pi}{2}} \frac{1}{1 - \cos\,\phi\,\cos\,\eta}\, d\eta = \frac{2}{\sin\phi} \arctan\Big(\frac{(1+\cos\phi)\tan(\frac{\eta}{2})}{\sin\phi}\Big)\Big|_{\eta=0}^{\frac{\pi}{2}}$$

$$= \frac{2}{\sin\phi} \arctan\Big(\frac{1+\cos\phi}{\sin\phi}\Big)$$

We also realize

$$
\begin{aligned}
&1+\cos\phi = 1+\cos 2(\frac{\phi}{2}) = 1+\cos^2(\frac{\phi}{2})-\sin^2(\frac{\phi}{2}) = 2\cos^2(\frac{\phi}{2}) \\
&\implies \cos(\frac{\phi}{2}) = \frac{1+\cos\phi}{2\cos(\frac{\phi}{2})} \\
&\implies \frac{\cos(\frac{\phi}{2})}{\sin(\frac{\phi}{2})} = \frac{1+\cos\phi}{2\sin(\frac{\phi}{2})\cos(\frac{\phi}{2})} = \frac{1+\cos\phi}{\sin\phi}
\end{aligned}
$$

Furthermore,

$$
\begin{aligned}
&\tan(\frac{\pi-\phi}{2}) = \frac{\sin(\frac{\pi}{2}-\frac{\phi}{2})}{\cos(\frac{\pi}{2}-\frac{\phi}{2})} = \frac{\cos(\frac{\phi}{2})}{\sin(\frac{\phi}{2})} = \frac{1+\cos\phi}{\sin\phi} \\
&\implies \frac{\pi-\phi}{\sin\phi} = \frac{2}{\sin\phi}\arctan\Big(\frac{1+\cos\phi}{\sin\phi}\Big) \\
&\implies \frac{\pi-\phi}{\sin\phi} = \int_{\eta=0}^{\frac{\pi}{2}} \frac{1}{1-\ \cos\ \phi\ \cos\ \eta}\, d\eta \qquad \blacksquare
\end{aligned}
$$

A.2 Covariance Function for Multi-hidden-layer ReLU Neural Networks

I wish to show that the exact recursive ReLU covariance function is given by

$$
\begin{aligned}
k^{(l)}(x,y) &= \frac{1}{2}\Big(\sigma_b^2 + k^{(l-1)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + k^{(l-1)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\xi^{(l)}(x,y) \qquad \text{(A.11)} \\
\text{with } \xi^{(l)}(x,y) &= \frac{1}{\pi}\Big(\sin\ \theta^{(l)}(x,y) + (\pi-\theta^{(l)}(x,y))\ \cos\ \theta^{(l)}(x,y)\Big), \\
\text{and } \theta^{(l)}(x,y) &= \cos^{-1}\left\{\frac{\sigma_b^2 + k^{(l-1)}(x,y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + k^{(l-1)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + k^{(l-1)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}\right\}.
\end{aligned}
$$

In order to evaluate the recursive ReLU covariance function, I need to apply Mercer's theorem and construct appropriate feature space mapping functions to establish the recursion.

As stated in [37], Mercer's theorem indicates that a covariance function can be represented as an inner product of feature vectors:

$$
k(x,y) = \big\langle \Phi(x), \Phi(y)\big\rangle = \|\Phi(x)\|\|\Phi(y)\|\ \cos\ \xi(x,y)
$$

where $\|\Phi(x)\| \coloneqq \sqrt{\big\langle \Phi(x), \Phi(y)\big\rangle}$ and $\xi(x,y)$ is the angle between $\Phi(x)$ and $\Phi(y)$.

I now illustrate the procedure for the first two layers. Then mathematical induction is applied to prove the claim for the general case.

Similar to the approach applied in [11, §2.3], I construct ReLU covariance functions corresponding to $l-$layer nested feature maps evaluated at the input $x, y \in \mathcal{R}^{d_{in}}$ as

$$k^{(l)}(x,y) = \big\langle \underbrace{\Phi(\Phi(\cdots\Phi(x))\cdots)}_{l \text{ terms}}, \underbrace{\Phi(\Phi(\cdots\Phi(y))\cdots)}_{l \text{ terms}} \big\rangle. \tag{A.12}$$

Recall that the covariance function at the output of the post-activation in the first hidden layer for inputs x, y is computed as

$$\begin{aligned}
&E[X_j^1(x)X_j^1(y)] = k^1(x,y) \\
&= \frac{1}{2\pi}\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sin\,\theta + (\pi-\theta)\,\cos\,\theta\Big), \\
&\text{where } \theta = \cos^{-1}\Bigg\{\frac{\sigma_b^2 + (x\cdot y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}.\Bigg\}
\end{aligned} \tag{A.13}$$

As a result, at the first layer I have

$$\begin{aligned}
k^{(1)}(x,x) &= \big\langle \Phi(x), \Phi(x)\big\rangle = \|\Phi(x)\|^2 = \frac{1}{2}\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big), \\
k^{(1)}(y,y) &= \big\langle \Phi(y), \Phi(y)\big\rangle = \|\Phi(y)\|^2 = \frac{1}{2}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big), \\
k^{(1)}(x,y) &= \big\langle \Phi(x), \Phi(y)\big\rangle = \|\Phi(x)\|\|\Phi(y)\|\,\xi^{(1)}(x,y) \\
&= \frac{1}{2}\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\xi^{(1)}(x,y), \\
\text{where } \xi^{(1)}(x,y) &= \frac{1}{\pi}\Big(\sin\,\theta^{(1)}(x,y) + (\pi - \theta^{(1)}(x,y))\,\cos\,\theta^{(1)}(x,y)\Big), \\
\text{and } \theta^{(1)}(x,y) &= \cos^{-1}\Bigg\{\frac{\sigma_b^2 + (x\cdot y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + \|x\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + \|y\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}\Bigg\}.
\end{aligned}$$

Applying the multilayer model (equation A.12), the variance at the second layer for input x becomes

$$\begin{aligned}
k^{(2)}(x,x) &= k^{(1)}(\Phi(x), \Phi(x)) \\
&= \frac{1}{2}\Big(\sigma_b^2 + \|\Phi(x)\|^2\frac{\sigma_w^2}{d_{in}}\Big)
\end{aligned}$$

$$= \frac{1}{2}\Big(\sigma_b^2 + k^{(1)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)$$

The ReLU covariance function at the second layer ($l = 2$) can therefore be evaluated in terms of the initial layer function:

$$\begin{aligned}
k^{(2)}(x,y) &= \big\langle \Phi(\Phi(x)), \Phi(\Phi(y)) \big\rangle \\
&= k^{(1)}(\Phi(x), \Phi(y)) \\
&= \frac{1}{2}\Big(\sigma_b^2 + \|\Phi(x)\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + \|\Phi(y)\|^2\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\xi^{(1)}(\Phi(x),\Phi(y)) \\
&= \frac{1}{2}\Big(\sigma_b^2 + k^{(1)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + k^{(1)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\xi^{(2)}(x,y)
\end{aligned}$$

$$\begin{aligned}
\text{with } \xi^{(2)}(x,y) &\coloneqq \xi^{(1)}(\Phi(x),\Phi(y)) \\
&= \frac{1}{\pi}\Big(\sin\,\theta^{(1)}(\Phi(x),\Phi(y)) + (\pi - \theta^{(1)}\Phi(x),\Phi(y))\,\cos\,\theta^{(1)}(\Phi(x),\Phi(y))\Big) \\
&= \frac{1}{\pi}\Big(\sin\,\theta^{(2)}(x,y) + (\pi - \theta^{(2)}(x,y))\,\cos\,\theta^{(2)}(x,y)\Big),
\end{aligned}$$

$$\begin{aligned}
\text{and } \theta^{(2)}(x,y) &\coloneqq \theta^{(1)}(\Phi(x),\Phi(y)) \\
&= \cos^{-1}\left\{\frac{\sigma_b^2 + \big(\Phi(x)\cdot\Phi(y)\big)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + k^{(1)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + k^{(1)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}\right\} \\
&= \cos^{-1}\left\{\frac{\sigma_b^2 + k^{(1)}(x,y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + k^{(1)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + k^{(1)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}}\right\}.
\end{aligned}$$

The procedure so far proves the recursion for $l = 2$. Now I show that if the recursion holds for any $l = m \in \mathbb{N}$, then it has to hold for the next case $l = m + 1$. Jointly, the two steps prove that the recursion holds for all natural numbers.

Proof of claim using mathematical induction

By definition,

$$\begin{aligned}
k^{(l+1)}(x,y) &= k^{(l)}(\|\Phi(x)\|, \|\Phi(y)\|) \\
&= \frac{1}{2}\Big(\sigma_b^2 + k^{(l-1)}(\|\Phi(x)\|,\|\Phi(x)\|)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + k^{(l)-1}(\|\Phi(y)\|,\|\Phi(y)\|)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}
\end{aligned}$$

$$
\begin{aligned}
&\xi^{(l)}(\|\Phi(x)\|, \|\Phi(y)\|) \\
&= \frac{1}{2}\Big(\sigma_b^2 + k^{(l)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\Big(\sigma_b^2 + k^{(l)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{\frac{1}{2}}\xi^{(l+1)}(x,y), \\
\text{with } \xi^{(l+1)}(x,y) &\coloneqq \xi^{(l)}(\Phi(x),\Phi(y)) \\
&= \frac{1}{\pi}\Big(\sin\,\theta^{(l)}(\Phi(x),\Phi(y)) + (\pi - \theta^{(l)}\Phi(x),\Phi(y))\,\cos\,\theta^{(l)}(\Phi(x),\Phi(y))\Big) \\
&= \frac{1}{\pi}\Big(\sin\,\theta^{(l+1)}(x,y) + (\pi - \theta^{(l+1)}(x,y))\,\cos\,\theta^{(l+1)}(x,y)\Big), \text{and} \\
\cos\Big(\theta^{(l+1)}(x,y)\Big) &\coloneqq \cos\Big(\theta^{(l)}(\Phi(x),\Phi(y))\Big) \\
&= \frac{\sigma_b^2 + k^{(l-1)}(\|\Phi(x)\|, \|\Phi(y)\|)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + k^{(l-1)}(\|\Phi(x)\|, \|\Phi(x)\|)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + k^{(l-1)}(\|\Phi(y)\|, \|\Phi(y)\|)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}} \\
&= \frac{\sigma_b^2 + k^{(l)}(x,y)\frac{\sigma_w^2}{d_{in}}}{\Big(\sigma_b^2 + k^{(l)}(x,x)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}\Big(\sigma_b^2 + k^{(l)}(y,y)\frac{\sigma_w^2}{d_{in}}\Big)^{1/2}} \qquad \blacksquare
\end{aligned}
$$

Therefore, by mathematical induction I prove that the recursive covariance function corresponding to a multilayer ReLU neural network model is given by equation

Anthony S. Tai

CONTACT INFORMATION

Myles Brand Hall 013
Department of Statistics
Indiana University
Bloomington, IN 47408

E-mail: astai@iu.edu
Homepage: https://astai.pages.iu.edu/

RESEARCH INTERESTS

Statistical machine learning, deep learning, Gaussian processes, spatial statistics, computer vision and pattern recognition, robust statistical modelling for artificial intelligence applications

EDUCATION

Indiana University, Bloomington, Indiana USA

Ph.D., Statistical Science, May 2021
Dissertation: *Initialization Strategy and Activation Function Selection for Neural Networks based on Gaussian Process Optimization*
Advisor: Dr. Chunfeng Huang

Indiana University, Bloomington, Indiana USA

Master of Science, Statistical Science, May 2016

Purdue University, West Lafayette, Indiana USA

Master of Science, Management, May 1992

Master of Science, Electrical Engineering, December 1986

University of Illinois, Urbana-Champaign, Illinois USA

Bachelor of Science, Electrical Engineering, July 1985

EMPLOYMENT HISTORY

Naval Surface Warfare Center - Crane Division, Crane, Indiana USA

Engineer **August 2009 - present**
Participated in research, design, and development of algorithms for signal emitter detection, identification, clustering and classification.

SAIC, Crane, Indiana USA

Senior RF Engineer **July 2008 - July 2009**
Provided engineering support to the TE2 Laboratory at NSWC, CRANE Division.

T-Mobile USA, Cincinnati, Ohio USA

RF Manager 2 **Octoer 2007 - June 2008**
Supervised Radio Frequency team in designing, optimizing GSM(Global System for Mobile Communications) network in Cincinnati, Ohio.

T-Mobile USA, Indianapolis, Indiana USA

Senior RF Engineer **January 1999 - September 2007**
Led Radio Frequency team in designing, optimizing, and maintaining NORTEL GSM networks in Indianapolis and Fort Wayne, Indiana.

Sprint PCS, Cincinnati, Ohio USA

Senior RF Engineer **July 1997 - December 1998**

Led Radio Frequency team in designing and deploying a MOTOROLA CDMA(Code Division Multiple Access, IS-95) network in Cincinnati, Ohio

NORTEL, Indianapolis, Indiana USA

Senior RF Engineer **May 1996 - June 1997**

Led Radio Frequency team in designing and deploying a MOTOROLA CDMA(Code Division Multiple Access, IS-95) network in Indianapolis, Indiana.

SAFCO Corporation, Chicago, Illinois USA

RF Engineer **October 1993 - April 1996**

Radio Frequency engineer supporting cellular system monitoring hardware and data analysis software testing and implementation

Honors and Awards

Naval Surface Warfare Center - Crane Division Ph.D. Fellowship, 2018-2021

Naval Warfare Center Pipper Innovation Award, Naval Sea Systems Command, U.S. Navy, 2012

Certification

Level III DAWIA(Defense Acquisition Workforce Improvement Act) Certification in Systems Planning, Research, Development, and Engineering – Systems Engineer(Science & Technology)

Computer Skills

Languages: Python, R, MatLab
Open-source libraries: Scikit-learn (Machine Learning), TensorFlow (Deep Learning), PIL (Imaging)
Applications: LaTeX, common Windows database, spreadsheet, and presentation software
Operating Systems: Windows, Unix/Linux.

Professional Society

The Institute of Electrical and Electronics Engineers (IEEE), 1992 - present

Publications

Anthony S. Tai and Chunfeng Huang. Guiding Neural Network Initialization via Marginal Likelihood Maximization. arXiv:2012.09943 [cs, stat], December 2020. arXiv: 2012.09943.